

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	215
5.23.2 Function Documentation	215
5.24 met_map.c	218
5.25 met_prof.c File Reference	221
5.25.1 Detailed Description	221
5.25.2 Function Documentation	221
5.26 met_prof.c	224
5.27 met_sample.c File Reference	227
5.27.1 Detailed Description	227
5.27.2 Function Documentation	227
5.28 met_sample.c	229
5.29 met_zm.c File Reference	232
5.29.1 Detailed Description	232
5.29.2 Function Documentation	232

5.30 met_zm.c	235
5.31 time2jsec.c File Reference	238
5.31.1 Detailed Description	238
5.31.2 Function Documentation	239
5.32 time2jsec.c	239
5.33 trac.c File Reference	240
5.33.1 Detailed Description	241
5.33.2 Function Documentation	241
5.33.3 Variable Documentation	262
5.34 trac.c	262
5.35 tropo.c File Reference	279
5.35.1 Detailed Description	279
5.35.2 Function Documentation	279
5.36 tropo.c	284
5.37 tropo_sample.c File Reference	288
5.37.1 Detailed Description	288
5.37.2 Function Documentation	288
5.38 tropo_sample.c	292
Index	297

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	221

met_sample.c	Sample meteorological data at given geolocations	227
met_zm.c	Extract zonal mean from meteorological data	232
time2jsec.c	Convert date to Julian seconds	238
trac.c	Lagrangian particle dispersion model	240
tropo.c	Create tropopause climatology from meteorological data	279
tropo_sample.c	Sample tropopause climatology	288

4 Data Structure Documentation

4.1 atm_t Struct Reference

Atmospheric data.

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

Data Fields

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

4.1.1 Detailed Description

Atmospheric data.

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

4.1.2 Field Documentation

4.1.2.1 `int atm_t::np`

Number of air parcels.

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

4.1.2.2 `double atm_t::time[NP]`

Time [s].

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

4.1.2.3 `double atm_t::p[NP]`

Pressure [hPa].

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

4.1.2.4 `double atm_t::lon[NP]`

Longitude [deg].

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

4.1.2.5 `double atm_t::lat[NP]`

Latitude [deg].

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

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

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

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

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

- [libtrac.h](#)

4.2 `cache_t` Struct Reference

Cache data.

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


Data Fields

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

4.2.1 Detailed Description

Cache data.

Definition at line 741 of file `libtrac.h`.

4.2.2 Field Documentation

4.2.2.1 float `cache_t::up`[NP]

Zonal wind perturbation [m/s].

Definition at line 744 of file `libtrac.h`.

4.2.2.2 float `cache_t::vp`[NP]

Meridional wind perturbation [m/s].

Definition at line 747 of file `libtrac.h`.

4.2.2.3 float `cache_t::wp`[NP]

Vertical velocity perturbation [hPa/s].

Definition at line 750 of file `libtrac.h`.

4.2.2.4 double cache_t::iso_var[NP]

Isosurface variables.

Definition at line 753 of file [libtrac.h](#).

4.2.2.5 double cache_t::iso_ps[NP]

Isosurface balloon pressure [hPa].

Definition at line 756 of file [libtrac.h](#).

4.2.2.6 double cache_t::iso_ts[NP]

Isosurface balloon time [s].

Definition at line 759 of file [libtrac.h](#).

4.2.2.7 int cache_t::iso_n

Isosurface balloon number of data points.

Definition at line 762 of file [libtrac.h](#).

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

Cache for reference time of wind standard deviations.

Definition at line 765 of file [libtrac.h](#).

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

Cache for zonal wind standard deviations.

Definition at line 768 of file [libtrac.h](#).

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

Cache for meridional wind standard deviations.

Definition at line 771 of file [libtrac.h](#).

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

Cache for vertical velocity standard deviations.

Definition at line 774 of file [libtrac.h](#).

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

- [libtrac.h](#)

4.3 `ctl_t` Struct Reference

Control parameters.

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

Data Fields

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

- int [qnt_hno3](#)
Quantity array index for nitric acid vmr.
- int [qnt_oh](#)
Quantity array index for hydroxyl number concentrations.
- int [qnt_rh](#)
Quantity array index for relative 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]

Basename of atmospheric data files.
- char `atm_gpfile` [LEN]

Gnuplot file for atmospheric data.
- double `atm_dt_out`

Time step for atmospheric data output [s].
- int `atm_filter`

Time filter for atmospheric data output (0=no, 1=yes).
- int `atm_stride`

Particle index stride for atmospheric data files.
- int `atm_type`

Type of atmospheric data files (0=ASCII, 1=binary, 2=netCDF).
- char `csi_basename` [LEN]

Basename of CSI data files.

- double [csi_dt_out](#)
Time step for CSI data output [s].
- char [csi_obsfile](#) [LEN]
Observation data file for CSI analysis.
- double [csi_obsmin](#)
Minimum observation index to trigger detection.
- double [csi_modmin](#)
Minimum column density to trigger detection [kg/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 369 of file `libtrac.h`.

4.3.2 Field Documentation

4.3.2.1 int `ctl_t::nq`

Number of quantities.

Definition at line 372 of file `libtrac.h`.

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

Quantity names.

Definition at line 375 of file [libtrac.h](#).

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

Quantity units.

Definition at line 378 of file [libtrac.h](#).

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

Quantity output format.

Definition at line 381 of file [libtrac.h](#).

4.3.2.5 int ctl_t::qnt_ens

Quantity array index for ensemble IDs.

Definition at line 384 of file [libtrac.h](#).

4.3.2.6 int ctl_t::qnt_m

Quantity array index for mass.

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

4.3.2.7 int ctl_t::qnt_rho

Quantity array index for particle density.

Definition at line 390 of file [libtrac.h](#).

4.3.2.8 int ctl_t::qnt_r

Quantity array index for particle radius.

Definition at line 393 of file [libtrac.h](#).

4.3.2.9 int ctl_t::qnt_ps

Quantity array index for surface pressure.

Definition at line 396 of file [libtrac.h](#).

4.3.2.10 int ctl_t::qnt_pt

Quantity array index for tropopause pressure.

Definition at line 399 of file [libtrac.h](#).

4.3.2.11 `int ctl_t::qnt_z`

Quantity array index for geopotential height.

Definition at line 402 of file [libtrac.h](#).

4.3.2.12 `int ctl_t::qnt_p`

Quantity array index for pressure.

Definition at line 405 of file [libtrac.h](#).

4.3.2.13 `int ctl_t::qnt_t`

Quantity array index for temperature.

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

4.3.2.14 `int ctl_t::qnt_u`

Quantity array index for zonal wind.

Definition at line 411 of file [libtrac.h](#).

4.3.2.15 `int ctl_t::qnt_v`

Quantity array index for meridional wind.

Definition at line 414 of file [libtrac.h](#).

4.3.2.16 `int ctl_t::qnt_w`

Quantity array index for vertical velocity.

Definition at line 417 of file [libtrac.h](#).

4.3.2.17 `int ctl_t::qnt_h2o`

Quantity array index for water vapor vmr.

Definition at line 420 of file [libtrac.h](#).

4.3.2.18 `int ctl_t::qnt_o3`

Quantity array index for ozone vmr.

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

4.3.2.19 `int ctl_t::qnt_lwc`

Quantity array index for cloud liquid water content.

Definition at line 426 of file [libtrac.h](#).

4.3.2.20 `int ctl_t::qnt_iwc`

Quantity array index for cloud ice water content.

Definition at line 429 of file [libtrac.h](#).

4.3.2.21 `int ctl_t::qnt_pc`

Quantity array index for cloud top pressure.

Definition at line 432 of file [libtrac.h](#).

4.3.2.22 `int ctl_t::qnt_hno3`

Quantity array index for nitric acid vmr.

Definition at line 435 of file [libtrac.h](#).

4.3.2.23 `int ctl_t::qnt_oh`

Quantity array index for hydroxyl number concentrations.

Definition at line 438 of file [libtrac.h](#).

4.3.2.24 `int ctl_t::qnt_rh`

Quantity array index for relative humidity.

Definition at line 441 of file [libtrac.h](#).

4.3.2.25 `int ctl_t::qnt_theta`

Quantity array index for potential temperature.

Definition at line 444 of file [libtrac.h](#).

4.3.2.26 `int ctl_t::qnt_vh`

Quantity array index for horizontal wind.

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

4.3.2.27 `int ctl_t::qnt_vz`

Quantity array index for vertical velocity.

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

4.3.2.28 `int ctl_t::qnt_pv`

Quantity array index for potential vorticity.

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

4.3.2.29 `int ctl_t::qnt_tice`

Quantity array index for T_ice.

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

4.3.2.30 `int ctl_t::qnt_tsts`

Quantity array index for T_STS.

Definition at line 459 of file [libtrac.h](#).

4.3.2.31 `int ctl_t::qnt_tnat`

Quantity array index for T_NAT.

Definition at line 462 of file [libtrac.h](#).

4.3.2.32 `int ctl_t::qnt_stat`

Quantity array index for station flag.

Definition at line 465 of file [libtrac.h](#).

4.3.2.33 `int ctl_t::direction`

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

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

4.3.2.34 `double ctl_t::t_start`

Start time of simulation [s].

Definition at line 471 of file [libtrac.h](#).

4.3.2.35 `double ctl_t::t_stop`

Stop time of simulation [s].

Definition at line 474 of file [libtrac.h](#).

4.3.2.36 `double ctl_t::dt_mod`

Time step of simulation [s].

Definition at line 477 of file [libtrac.h](#).

4.3.2.37 `double ctl_t::dt_met`

Time step of meteorological data [s].

Definition at line 480 of file [libtrac.h](#).

4.3.2.38 `int ctl_t::met_dx`

Stride for longitudes.

Definition at line 483 of file [libtrac.h](#).

4.3.2.39 `int ctl_t::met_dy`

Stride for latitudes.

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

4.3.2.40 `int ctl_t::met_dp`

Stride for pressure levels.

Definition at line 489 of file [libtrac.h](#).

4.3.2.41 `int ctl_t::met_sx`

Smoothing for longitudes.

Definition at line 492 of file [libtrac.h](#).

4.3.2.42 `int ctl_t::met_sy`

Smoothing for latitudes.

Definition at line 495 of file [libtrac.h](#).

4.3.2.43 `int ctl_t::met_sp`

Smoothing for pressure levels.

Definition at line 498 of file [libtrac.h](#).

4.3.2.44 `int ctl_t::met_np`

Number of target pressure levels.

Definition at line 501 of file [libtrac.h](#).

4.3.2.45 `double ctl_t::met_p[EP]`

Target pressure levels [hPa].

Definition at line 504 of file [libtrac.h](#).

4.3.2.46 `int ctl_t::met_tropo`

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

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

4.3.2.47 `char ctl_t::met_geopot[LEN]`

Surface geopotential data file.

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

4.3.2.48 `double ctl_t::met_dt_out`

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

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

4.3.2.49 `char ctl_t::met_stage[LEN]`

Command to stage meteo data.

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

4.3.2.50 `int ctl_t::isosurf`

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

Definition at line 521 of file [libtrac.h](#).

4.3.2.51 `char ctl_t::balloon[LEN]`

Balloon position filename.

Definition at line 524 of file [libtrac.h](#).

4.3.2.52 `double ctl_t::turb_dx_trop`

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

Definition at line 527 of file [libtrac.h](#).

4.3.2.53 `double ctl_t::turb_dx_strat`

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

Definition at line 530 of file [libtrac.h](#).

4.3.2.54 `double ctl_t::turb_dz_trop`

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

Definition at line 533 of file [libtrac.h](#).

4.3.2.55 `double ctl_t::turb_dz_strat`

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

Definition at line 536 of file [libtrac.h](#).

4.3.2.56 double ctl_t::turb_mesox

Horizontal scaling factor for mesoscale wind fluctuations.

Definition at line 539 of file [libtrac.h](#).

4.3.2.57 double ctl_t::turb_mesoz

Vertical scaling factor for mesoscale wind fluctuations.

Definition at line 542 of file [libtrac.h](#).

4.3.2.58 char ctl_t::species[LEN]

Species.

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

4.3.2.59 double ctl_t::molmass

Molar mass [g/mol].

Definition at line 548 of file [libtrac.h](#).

4.3.2.60 double ctl_t::tdec_trop

Life time of particles (troposphere) [s].

Definition at line 551 of file [libtrac.h](#).

4.3.2.61 double ctl_t::tdec_strat

Life time of particles (stratosphere) [s].

Definition at line 554 of file [libtrac.h](#).

4.3.2.62 double ctl_t::oh_chem[4]

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

Definition at line 557 of file [libtrac.h](#).

4.3.2.63 double ctl_t::wet_depo[4]

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

Definition at line 560 of file [libtrac.h](#).

4.3.2.64 double ctl_t::psc_h2o

H2O volume mixing ratio for PSC analysis.

Definition at line 563 of file [libtrac.h](#).

4.3.2.65 `double ctl_t::psc_hno3`

HNO3 volume mixing ratio for PSC analysis.

Definition at line 566 of file [libtrac.h](#).

4.3.2.66 `char ctl_t::atm_basename[LEN]`

Basename of atmospheric data files.

Definition at line 569 of file [libtrac.h](#).

4.3.2.67 `char ctl_t::atm_gpfile[LEN]`

Gnuplot file for atmospheric data.

Definition at line 572 of file [libtrac.h](#).

4.3.2.68 `double ctl_t::atm_dt_out`

Time step for atmospheric data output [s].

Definition at line 575 of file [libtrac.h](#).

4.3.2.69 `int ctl_t::atm_filter`

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

Definition at line 578 of file [libtrac.h](#).

4.3.2.70 `int ctl_t::atm_stride`

Particle index stride for atmospheric data files.

Definition at line 581 of file [libtrac.h](#).

4.3.2.71 `int ctl_t::atm_type`

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

Definition at line 584 of file [libtrac.h](#).

4.3.2.72 `char ctl_t::csi_basename[LEN]`

Basename of CSI data files.

Definition at line 587 of file [libtrac.h](#).

4.3.2.73 `double ctl_t::csi_dt_out`

Time step for CSI data output [s].

Definition at line 590 of file [libtrac.h](#).

4.3.2.74 `char ctl_t::csi_obsfile[LEN]`

Observation data file for CSI analysis.

Definition at line 593 of file [libtrac.h](#).

4.3.2.75 `double ctl_t::csi_obsmin`

Minimum observation index to trigger detection.

Definition at line 596 of file [libtrac.h](#).

4.3.2.76 `double ctl_t::csi_modmin`

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

Definition at line 599 of file [libtrac.h](#).

4.3.2.77 `int ctl_t::csi_nz`

Number of altitudes of gridded CSI data.

Definition at line 602 of file [libtrac.h](#).

4.3.2.78 `double ctl_t::csi_z0`

Lower altitude of gridded CSI data [km].

Definition at line 605 of file [libtrac.h](#).

4.3.2.79 `double ctl_t::csi_z1`

Upper altitude of gridded CSI data [km].

Definition at line 608 of file [libtrac.h](#).

4.3.2.80 `int ctl_t::csi_nx`

Number of longitudes of gridded CSI data.

Definition at line 611 of file [libtrac.h](#).

4.3.2.81 `double ctl_t::csi_lon0`

Lower longitude of gridded CSI data [deg].

Definition at line 614 of file [libtrac.h](#).

4.3.2.82 `double ctl_t::csi_lon1`

Upper longitude of gridded CSI data [deg].

Definition at line 617 of file [libtrac.h](#).

4.3.2.83 `int ctl_t::csi_ny`

Number of latitudes of gridded CSI data.

Definition at line 620 of file [libtrac.h](#).

4.3.2.84 `double ctl_t::csi_lat0`

Lower latitude of gridded CSI data [deg].

Definition at line 623 of file [libtrac.h](#).

4.3.2.85 `double ctl_t::csi_lat1`

Upper latitude of gridded CSI data [deg].

Definition at line 626 of file [libtrac.h](#).

4.3.2.86 `char ctl_t::grid_basename[LEN]`

Basename of grid data files.

Definition at line 629 of file [libtrac.h](#).

4.3.2.87 `char ctl_t::grid_gfile[LEN]`

Gnuplot file for gridded data.

Definition at line 632 of file [libtrac.h](#).

4.3.2.88 `double ctl_t::grid_dt_out`

Time step for gridded data output [s].

Definition at line 635 of file [libtrac.h](#).

4.3.2.89 `int ctl_t::grid_sparse`

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

Definition at line 638 of file [libtrac.h](#).

4.3.2.90 `int ctl_t::grid_nz`

Number of altitudes of gridded data.

Definition at line 641 of file [libtrac.h](#).

4.3.2.91 `double ctl_t::grid_z0`

Lower altitude of gridded data [km].

Definition at line 644 of file [libtrac.h](#).

4.3.2.92 double ctl_t::grid_z1

Upper altitude of gridded data [km].

Definition at line 647 of file [libtrac.h](#).

4.3.2.93 int ctl_t::grid_nx

Number of longitudes of gridded data.

Definition at line 650 of file [libtrac.h](#).

4.3.2.94 double ctl_t::grid_lon0

Lower longitude of gridded data [deg].

Definition at line 653 of file [libtrac.h](#).

4.3.2.95 double ctl_t::grid_lon1

Upper longitude of gridded data [deg].

Definition at line 656 of file [libtrac.h](#).

4.3.2.96 int ctl_t::grid_ny

Number of latitudes of gridded data.

Definition at line 659 of file [libtrac.h](#).

4.3.2.97 double ctl_t::grid_lat0

Lower latitude of gridded data [deg].

Definition at line 662 of file [libtrac.h](#).

4.3.2.98 double ctl_t::grid_lat1

Upper latitude of gridded data [deg].

Definition at line 665 of file [libtrac.h](#).

4.3.2.99 char ctl_t::prof_basename[LEN]

Basename for profile output file.

Definition at line 668 of file [libtrac.h](#).

4.3.2.100 char ctl_t::prof_obsfile[LEN]

Observation data file for profile output.

Definition at line 671 of file [libtrac.h](#).

4.3.2.101 `int ctl_t::prof_nz`

Number of altitudes of gridded profile data.

Definition at line 674 of file [libtrac.h](#).

4.3.2.102 `double ctl_t::prof_z0`

Lower altitude of gridded profile data [km].

Definition at line 677 of file [libtrac.h](#).

4.3.2.103 `double ctl_t::prof_z1`

Upper altitude of gridded profile data [km].

Definition at line 680 of file [libtrac.h](#).

4.3.2.104 `int ctl_t::prof_nx`

Number of longitudes of gridded profile data.

Definition at line 683 of file [libtrac.h](#).

4.3.2.105 `double ctl_t::prof_lon0`

Lower longitude of gridded profile data [deg].

Definition at line 686 of file [libtrac.h](#).

4.3.2.106 `double ctl_t::prof_lon1`

Upper longitude of gridded profile data [deg].

Definition at line 689 of file [libtrac.h](#).

4.3.2.107 `int ctl_t::prof_ny`

Number of latitudes of gridded profile data.

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

4.3.2.108 `double ctl_t::prof_lat0`

Lower latitude of gridded profile data [deg].

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

4.3.2.109 `double ctl_t::prof_lat1`

Upper latitude of gridded profile data [deg].

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

4.3.2.110 `char ctl_t::ens_basename[LEN]`

Basename of ensemble data file.

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

4.3.2.111 `char ctl_t::stat_basename[LEN]`

Basename of station data file.

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

4.3.2.112 `double ctl_t::stat_lon`

Longitude of station [deg].

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

4.3.2.113 `double ctl_t::stat_lat`

Latitude of station [deg].

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

4.3.2.114 `double ctl_t::stat_r`

Search radius around station [km].

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

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

- [libtrac.h](#)

4.4 `met_t` Struct Reference

Meteorological data.

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

Data Fields

- double [time](#)
Time [s].
- int [nx](#)
Number of longitudes.
- int [ny](#)
Number of latitudes.
- int [np](#)
Number of pressure levels.
- double [lon](#) [EX]
Longitude [deg].
- double [lat](#) [EY]
Latitude [deg].
- double [p](#) [EP]
Pressure [hPa].
- float [ps](#) [EX][EY]
Surface pressure [hPa].
- float [zs](#) [EX][EY]
Geopotential height at the surface [km].
- float [pt](#) [EX][EY]
Tropopause pressure [hPa].
- float [pc](#) [EX][EY]
Cloud top pressure [hPa].
- float [cl](#) [EX][EY]
Total column cloud water [kg/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 779 of file [libtrac.h](#).

4.4.2 Field Documentation

4.4.2.1 double met_t::time

Time [s].

Definition at line 782 of file [libtrac.h](#).

4.4.2.2 int met_t::nx

Number of longitudes.

Definition at line 785 of file [libtrac.h](#).

4.4.2.3 int met_t::ny

Number of latitudes.

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

4.4.2.4 int met_t::np

Number of pressure levels.

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

4.4.2.5 double met_t::lon[EX]

Longitude [deg].

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

4.4.2.6 double met_t::lat[EY]

Latitude [deg].

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

4.4.2.7 double met_t::p[EP]

Pressure [hPa].

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

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

Surface pressure [hPa].

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

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

Geopotential height at the surface [km].

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

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

Tropopause pressure [hPa].

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

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

Cloud top pressure [hPa].

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

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

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

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

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

Geopotential height at model levels [km].

Definition at line 818 of file [libtrac.h](#).

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

Temperature [K].

Definition at line 821 of file [libtrac.h](#).

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

Zonal wind [m/s].

Definition at line 824 of file [libtrac.h](#).

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

Meridional wind [m/s].

Definition at line 827 of file [libtrac.h](#).

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

Vertical wind [hPa/s].

Definition at line 830 of file [libtrac.h](#).

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

Potential vorticity [PVU].

Definition at line 833 of file [libtrac.h](#).

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

Water vapor volume mixing ratio [1].

Definition at line 836 of file [libtrac.h](#).

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

Ozone volume mixing ratio [1].

Definition at line 839 of file [libtrac.h](#).

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

Cloud liquid water content [kg/kg].

Definition at line 842 of file [libtrac.h](#).

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

Cloud ice water content [kg/kg].

Definition at line 845 of file [libtrac.h](#).

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

Pressure on model levels [hPa].

Definition at line 848 of file [libtrac.h](#).

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

- [libtrac.h](#)

5 File Documentation

5.1 atm_conv.c File Reference

Convert file format of air parcel data files.

Functions

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

5.1.1 Detailed Description

Convert file format of air parcel data files.

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

5.1.2 Function Documentation

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

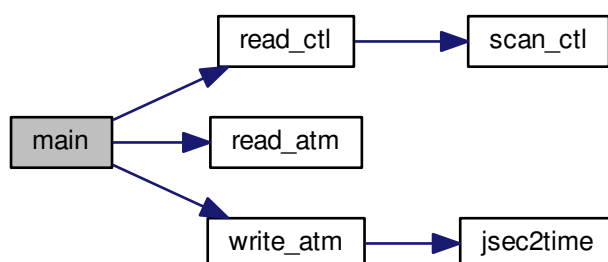
Definition at line 27 of file [atm_conv.c](#).

```

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> <atm1a> <atm1b>"
00087             " [<atm2a> <atm2b> ...]");
00088
00089     /* Read control parameters... */
00090     read_ctl(argv[1], argc, argv, &ctl);
00091     ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-999", NULL);
00092     p0 = P(scan_ctl(argv[1], argc, argv, "DIST_Z0", -1, "-1000", NULL));
00093     p1 = P(scan_ctl(argv[1], argc, argv, "DIST_Z1", -1, "1000", NULL));
00094     lat0 = scan_ctl(argv[1], argc, argv, "DIST_LAT0", -1, "-1000", NULL);
00095     lat1 = scan_ctl(argv[1], argc, argv, "DIST_LAT1", -1, "1000", NULL);
00096     lon0 = scan_ctl(argv[1], argc, argv, "DIST_LON0", -1, "-1000", NULL);
00097     lon1 = scan_ctl(argv[1], argc, argv, "DIST_LON1", -1, "1000", NULL);
00098
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     }

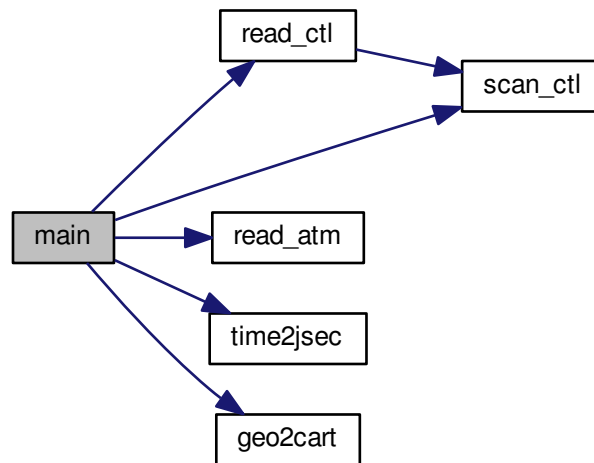
```

```

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             "# $qd = %s absolute difference (%s) [%s]\n"
00118             "# $qd = %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, "# $qd = 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... */

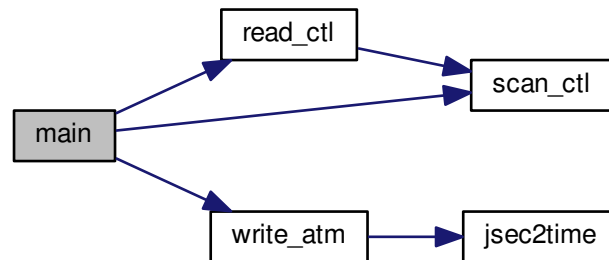
```

```

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_rng_gaussian_ziggurat(rng, st / 2.3548)
00090                            + ut * (gsl_rng_uniform(rng) - 0.5));
00091                     atm->p[atm->np]
00092                         = P(z + gsl_rng_gaussian_ziggurat(rng, sz / 2.3548)
00093                            + uz * (gsl_rng_uniform(rng) - 0.5));
00094                     atm->lon[atm->np]
00095                         = (lon + gsl_rng_gaussian_ziggurat(rng, slon / 2.3548)
00096                            + gsl_rng_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_rng_gaussian_ziggurat(rng, slat / 2.3548)
00101                                + gsl_rng_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

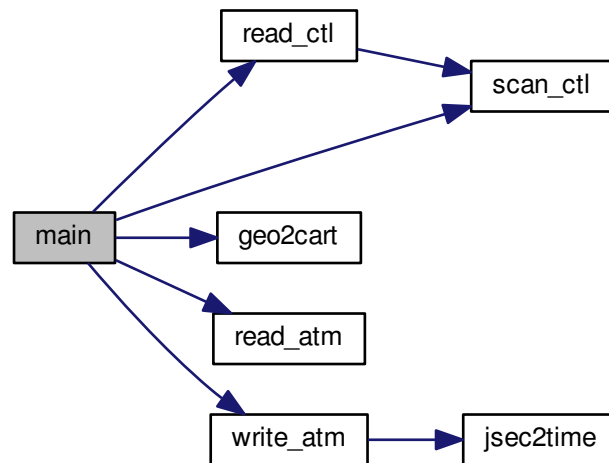
```

```

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);

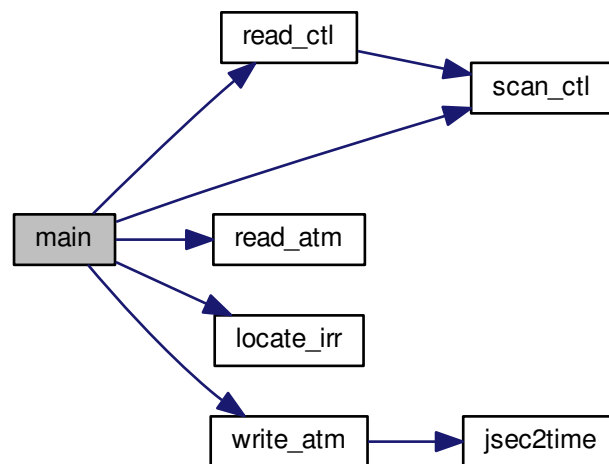
```

```

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);

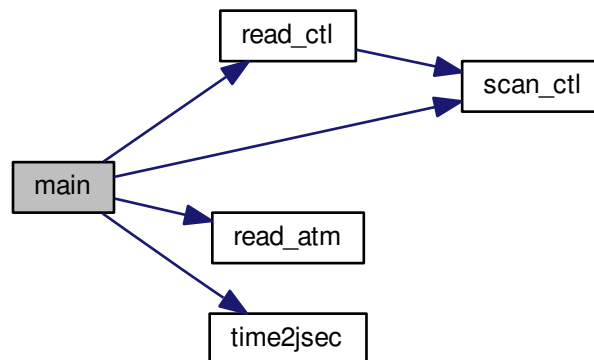
```

```

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 295 of file [libtrac.c](#).

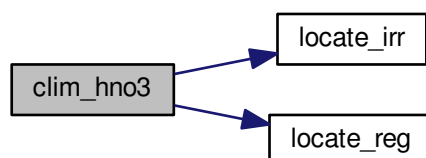
```
00298         {
00299
00300     /* Get seconds since begin of year... */
00301     double sec = FMOD(t, 365.25 * 86400.);
00302     while (sec < 0)
00303         sec += 365.25 * 86400.;
00304
00305     /* Check pressure... */
00306     if (p < clim_hno3_ps[0])
00307         p = clim_hno3_ps[0];
00308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

```

00329 double aux11 = LIN(clim_hno3_ps[ip],
00330                    clim_hno3_var[isec + 1][ilat + 1][ip],
00331                    clim_hno3_ps[ip + 1],
00332                    clim_hno3_var[isec + 1][ilat + 1][ip + 1], p);
00333 aux00 = LIN(clim_hno3_lats[ilat], aux00,
00334            clim_hno3_lats[ilat + 1], aux01, lat);
00335 aux11 = LIN(clim_hno3_lats[ilat], aux10,
00336            clim_hno3_lats[ilat + 1], aux11, lat);
00337 return LIN(clim_hno3_secs[isec], aux00,
00338            clim_hno3_secs[isec + 1], aux11, sec);
00339 }

```

Here is the call graph for this function:



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

Climatology of OH number concentrations.

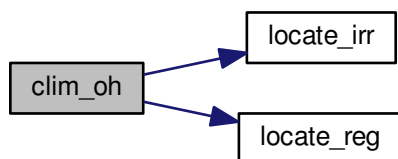
Definition at line 1322 of file [libtrac.c](#).

```

01325 {
01326
01327 /* Get seconds since begin of year... */
01328 double sec = FMOD(t, 365.25 * 86400.);
01329 while (sec < 0)
01330     sec += 365.25 * 86400.;
01331
01332 /* Check pressure... */
01333 if (p < clim_oh_ps[0])
01334     p = clim_oh_ps[0];
01335 else if (p > clim_oh_ps[33])
01336     p = clim_oh_ps[33];
01337
01338 /* Get indices... */
01339 int isec = locate_irr(clim_oh_secs, 12, sec);
01340 int ilat = locate_reg(clim_oh_lats, 18, lat);
01341 int ip = locate_irr(clim_oh_ps, 34, p);
01342
01343 /* Interpolate OH climatology (Pommrich et al., 2014)... */
01344 double aux00 = LIN(clim_oh_ps[ip],
01345                  clim_oh_var[isec][ilat][ip],
01346                  clim_oh_ps[ip + 1],
01347                  clim_oh_var[isec][ilat][ip + 1], p);
01348 double aux01 = LIN(clim_oh_ps[ip],
01349                  clim_oh_var[isec][ilat + 1][ip],
01350                  clim_oh_ps[ip + 1],
01351                  clim_oh_var[isec][ilat + 1][ip + 1], p);
01352 double aux10 = LIN(clim_oh_ps[ip],
01353                  clim_oh_var[isec + 1][ilat][ip],
01354                  clim_oh_ps[ip + 1],
01355                  clim_oh_var[isec + 1][ilat][ip + 1], p);
01356 double aux11 = LIN(clim_oh_ps[ip],
01357                  clim_oh_var[isec + 1][ilat + 1][ip],
01358                  clim_oh_ps[ip + 1],
01359                  clim_oh_var[isec + 1][ilat + 1][ip + 1], p);
01360 aux00 = LIN(clim_oh_lats[ilat], aux00, clim_oh_lats[ilat + 1], aux01, lat);
01361 aux11 = LIN(clim_oh_lats[ilat], aux10, clim_oh_lats[ilat + 1], aux11, lat);
01362 return 1e6 * LIN(clim_oh_secs[isec], aux00,
01363                clim_oh_secs[isec + 1], aux11, sec);
01364 }

```

Here is the call graph for this function:



5.19.2.4 double clim_tropo (double t, double lat)

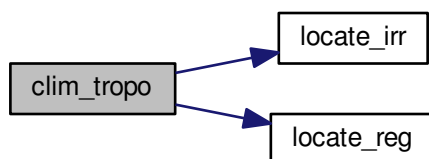
Climatology of tropopause pressure.

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

```

01499         {
01500
01501         /* Get seconds since begin of year... */
01502         double sec = FMOD(t, 365.25 * 86400.);
01503         while (sec < 0)
01504             sec += 365.25 * 86400.;
01505
01506         /* Get indices... */
01507         int isec = locate_irr(clim_tropo_secs, 12, sec);
01508         int ilat = locate_reg(clim_tropo_lats, 73, lat);
01509
01510         /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
01511         double p0 = LIN(clim_tropo_lats[ilat],
01512                        clim_tropo_tps[isec][ilat],
01513                        clim_tropo_lats[ilat + 1],
01514                        clim_tropo_tps[isec][ilat + 1], lat);
01515         double p1 = LIN(clim_tropo_lats[ilat],
01516                        clim_tropo_tps[isec + 1][ilat],
01517                        clim_tropo_lats[ilat + 1],
01518                        clim_tropo_tps[isec + 1][ilat + 1], lat);
01519         return LIN(clim_tropo_secs[isec], p0, clim_tropo_secs[isec + 1], p1, sec);
01520     }
  
```

Here is the call graph for this function:



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

Get day of year from date.

Definition at line 1524 of file libtrac.c.

```
01528     {
01529
01530     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01531     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01532
01533     /* Get day of year... */
01534     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
01535         *doy = d0l[mon - 1] + day - 1;
01536     else
01537         *doy = d0[mon - 1] + day - 1;
01538 }
```

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

Get date from day of year.

Definition at line 1542 of file libtrac.c.

```
01546     {
01547
01548     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01549     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01550     int i;
01551
01552     /* Get month and day... */
01553     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
01554         for (i = 11; i >= 0; i--)
01555             if (d0l[i] <= doy)
01556                 break;
01557         *mon = i + 1;
01558         *day = doy - d0l[i] + 1;
01559     } else {
01560         for (i = 11; i >= 0; i--)
01561             if (d0[i] <= doy)
01562                 break;
01563         *mon = i + 1;
01564         *day = doy - d0[i] + 1;
01565     }
01566 }
```

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

Convert geolocation to Cartesian coordinates.

Definition at line 1570 of file libtrac.c.

```
01574     {
01575
01576     double radius = z + RE;
01577     x[0] = radius * cos(lat / 180. * M_PI) * cos(lon / 180. * M_PI);
01578     x[1] = radius * cos(lat / 180. * M_PI) * sin(lon / 180. * M_PI);
01579     x[2] = radius * sin(lat / 180. * M_PI);
01580 }
```

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

Get meteorological data for given timestep.

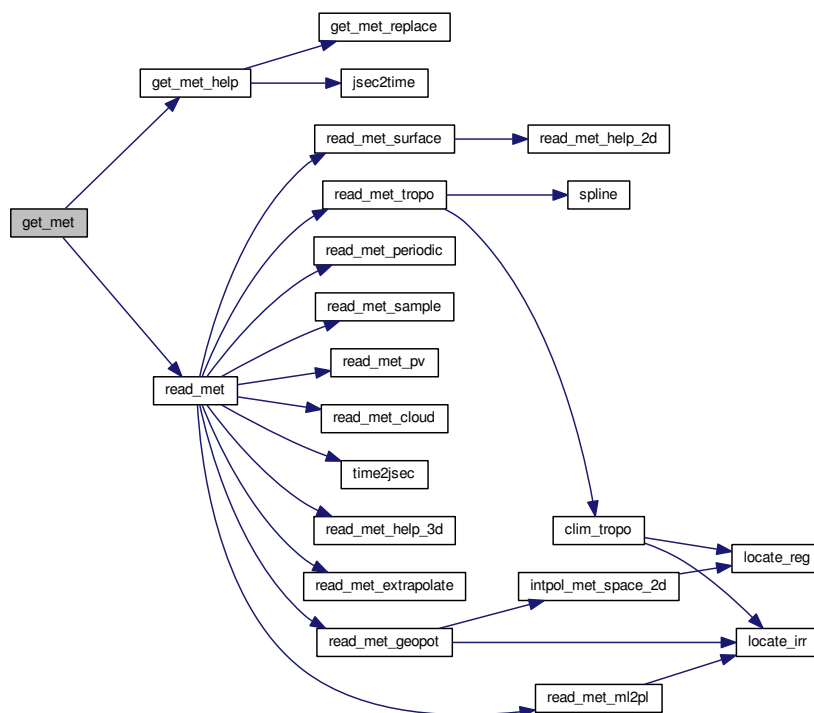
Definition at line 1584 of file libtrac.c.

```

01589         {
01590
01591     static int init, ip, ix, iy;
01592
01593     met_t *mets;
01594
01595     char filename[LEN];
01596
01597     /* Init... */
01598     if (t == ctl->t_start || !init) {
01599         init = 1;
01600
01601         get_met_help(t, -1, metbase, ctl->dt_met, filename);
01602         if (!read_met(ctl, filename, *met0))
01603             ERRMSG("Cannot open file!");
01604
01605         get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
dt_met, filename);
01606         if (!read_met(ctl, filename, *met1))
01607             ERRMSG("Cannot open file!");
01608 #ifdef _OPENACC
01609         met_t *met0up = *met0;
01610         met_t *met1up = *met1;
01611 #pragma acc update device(met0up[:1],met1up[:1])
01612 #endif
01613     }
01614
01615     /* Read new data for forward trajectories... */
01616     if (t > (*met1)->time && ctl->direction == 1) {
01617         mets = *met1;
01618         *met1 = *met0;
01619         *met0 = mets;
01620         get_met_help(t, 1, metbase, ctl->dt_met, filename);
01621         if (!read_met(ctl, filename, *met1))
01622             ERRMSG("Cannot open file!");
01623 #ifdef _OPENACC
01624         met_t *met1up = *met1;
01625 #pragma acc update device(met1up[:1])
01626 #endif
01627     }
01628
01629     /* Read new data for backward trajectories... */
01630     if (t < (*met0)->time && ctl->direction == -1) {
01631         mets = *met1;
01632         *met1 = *met0;
01633         *met0 = mets;
01634         get_met_help(t, -1, metbase, ctl->dt_met, filename);
01635         if (!read_met(ctl, filename, *met0))
01636             ERRMSG("Cannot open file!");
01637 #ifdef _OPENACC
01638         met_t *met0up = *met0;
01639 #pragma acc update device(met0up[:1])
01640 #endif
01641     }
01642
01643     /* Check that grids are consistent... */
01644     if (((*met0)->nx != (*met1)->nx
01645         || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
01646         ERRMSG("Meteo grid dimensions do not match!");
01647     for (ix = 0; ix < (*met0)->nx; ix++)
01648         if ((*met0)->lon[ix] != (*met1)->lon[ix])
01649             ERRMSG("Meteo grid longitudes do not match!");
01650     for (iy = 0; iy < (*met0)->ny; iy++)
01651         if ((*met0)->lat[iy] != (*met1)->lat[iy])
01652             ERRMSG("Meteo grid latitudes do not match!");
01653     for (ip = 0; ip < (*met0)->np; ip++)
01654         if ((*met0)->p[ip] != (*met1)->p[ip])
01655             ERRMSG("Meteo grid pressure levels do not match!");
01656 }

```


Here is the call graph for this function:



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

Get meteorological data for timestep.

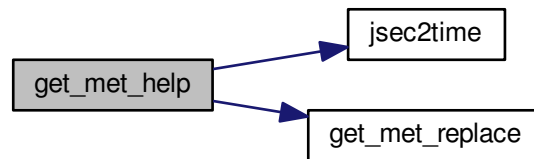
Definition at line 1660 of file libtrac.c.

```

01665         {
01666
01667     char repl[LEN];
01668
01669     double t6, r;
01670
01671     int year, mon, day, hour, min, sec;
01672
01673     /* Round time to fixed intervals... */
01674     if (direct == -1)
01675         t6 = floor(t / dt_met) * dt_met;
01676     else
01677         t6 = ceil(t / dt_met) * dt_met;
01678
01679     /* Decode time... */
01680     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
01681
01682     /* Set filename... */
01683     sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
01684     sprintf(repl, "%d", year);
01685     get_met_replace(filename, "YYYY", repl);
01686     sprintf(repl, "%02d", mon);
01687     get_met_replace(filename, "MM", repl);
01688     sprintf(repl, "%02d", day);
01689     get_met_replace(filename, "DD", repl);
01690     sprintf(repl, "%02d", hour);
01691     get_met_replace(filename, "HH", repl);
01692 }

```

Here is the call graph for this function:



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

Replace template strings in filename.

Definition at line 1696 of file [libtrac.c](#).

```

01699         {
01700
01701     char buffer[LEN], *ch;
01702
01703     /* Iterate... */
01704     for (int i = 0; i < 3; i++) {
01705
01706         /* Replace substring... */
01707         if (!(ch = strstr(orig, search)))
01708             return;
01709         strncpy(buffer, orig, (size_t) (ch - orig));
01710         buffer[ch - orig] = 0;
01711         sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
01712         orig[0] = 0;
01713         strcpy(orig, buffer);
01714     }
01715 }
  
```

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

Spatial interpolation of meteorological data.

Definition at line 1719 of file [libtrac.c](#).

```

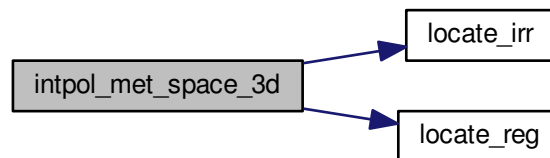
01728         {
01729
01730     /* Check longitude... */
01731     if (met->lon[met->nx - 1] > 180 && lon < 0)
01732         lon += 360;
01733
01734     /* Get interpolation indices and weights... */
01735     if (init) {
01736         ci[0] = locate_irr(met->p, met->np, p);
01737         ci[1] = locate_reg(met->lon, met->nx, lon);
01738         ci[2] = locate_reg(met->lat, met->ny, lat);
01739         cw[0] = (met->p[ci[0] + 1] - p)
01740             / (met->p[ci[0] + 1] - met->p[ci[0]]);
01741         cw[1] = (met->lon[ci[1] + 1] - lon)
01742             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01743         cw[2] = (met->lat[ci[2] + 1] - lat)
01744             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01745     }
01746
01747     /* Interpolate vertically... */
  
```

```

01748 double aux00 =
01749     cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
01750     + array[ci[1]][ci[2]][ci[0] + 1];
01751 double aux01 =
01752     cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
01753             array[ci[1]][ci[2] + 1][ci[0] + 1])
01754     + array[ci[1]][ci[2] + 1][ci[0] + 1];
01755 double aux10 =
01756     cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
01757             array[ci[1] + 1][ci[2]][ci[0] + 1])
01758     + array[ci[1] + 1][ci[2]][ci[0] + 1];
01759 double aux11 =
01760     cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
01761             array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
01762     + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];
01763
01764 /* Interpolate horizontally... */
01765 aux00 = cw[2] * (aux00 - aux01) + aux01;
01766 aux11 = cw[2] * (aux10 - aux11) + aux11;
01767 *var = cw[1] * (aux00 - aux11) + aux11;
01768 }

```

Here is the call graph for this function:



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

Spatial interpolation of meteorological data.

Definition at line 1773 of file `libtrac.c`.

```

01781     {
01782
01783     /* Check longitude... */
01784     if (met->lon[met->nx - 1] > 180 && lon < 0)
01785         lon += 360;
01786
01787     /* Get interpolation indices and weights... */
01788     if (init) {
01789         ci[1] = locate_reg(met->lon, met->nx, lon);
01790         ci[2] = locate_reg(met->lat, met->ny, lat);
01791         cw[1] = (met->lon[ci[1] + 1] - lon)
01792             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01793         cw[2] = (met->lat[ci[2] + 1] - lat)
01794             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01795     }
01796
01797     /* Set variables... */
01798     double aux00 = array[ci[1]][ci[2]];
01799     double aux01 = array[ci[1]][ci[2] + 1];
01800     double aux10 = array[ci[1] + 1][ci[2]];
01801     double aux11 = array[ci[1] + 1][ci[2] + 1];
01802
01803     /* Interpolate horizontally... */
01804     if (isfinite(aux00) && isfinite(aux01))
01805         aux00 = cw[2] * (aux00 - aux01) + aux01;
01806     else if (cw[2] < 0.5)

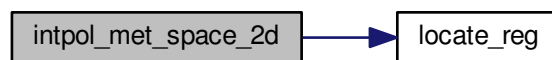
```

```

01807     aux00 = aux01;
01808     if (isfinite(aux10) && isfinite(aux11))
01809         aux11 = cw[2] * (aux10 - aux11) + aux11;
01810     else if (cw[2] > 0.5)
01811         aux11 = aux10;
01812     if (isfinite(aux00) && isfinite(aux11))
01813         *var = cw[1] * (aux00 - aux11) + aux11;
01814     else {
01815         if (cw[1] > 0.5)
01816             *var = aux00;
01817         else
01818             *var = aux11;
01819     }
01820 }

```

Here is the call graph for this function:



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

Temporal interpolation of meteorological data.

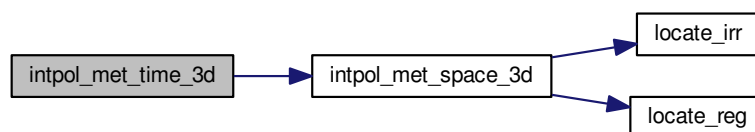
Definition at line 1824 of file `libtrac.c`.

```

01836     {
01837
01838         double var0, var1, wt;
01839
01840         /* Spatial interpolation... */
01841         intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
01842         intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, init);
01843
01844         /* Get weighting factor... */
01845         wt = (met1->time - ts) / (met1->time - met0->time);
01846
01847         /* Interpolate... */
01848         *var = wt * (var0 - var1) + var1;
01849     }

```

Here is the call graph for this function:



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

Temporal interpolation of meteorological data.

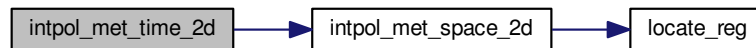
Definition at line 1853 of file `libtrac.c`.

```

01864         {
01865
01866     double var0, var1, wt;
01867
01868     /* Spatial interpolation... */
01869     intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
01870     intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, init);
01871
01872     /* Get weighting factor... */
01873     wt = (met1->time - ts) / (met1->time - met0->time);
01874
01875     /* Interpolate... */
01876     *var = wt * (var0 - var1) + var1;
01877 }

```

Here is the call graph for this function:



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

Convert seconds to date.

Definition at line 1881 of file `libtrac.c`.

```

01889         {
01890
01891     struct tm t0, *t1;
01892
01893     t0.tm_year = 100;
01894     t0.tm_mon = 0;
01895     t0.tm_mday = 1;
01896     t0.tm_hour = 0;
01897     t0.tm_min = 0;
01898     t0.tm_sec = 0;
01899
01900     time_t jsec0 = (time_t) jsec + timegm(&t0);
01901     t1 = gmtime(&jsec0);
01902
01903     *year = t1->tm_year + 1900;
01904     *mon = t1->tm_mon + 1;
01905     *day = t1->tm_mday;
01906     *hour = t1->tm_hour;
01907     *min = t1->tm_min;
01908     *sec = t1->tm_sec;
01909     *remain = jsec - floor(jsec);
01910 }

```

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

Find array index for irregular grid.

Definition at line 1914 of file libtrac.c.

```

01917         {
01918
01919     int ilo = 0;
01920     int ihi = n - 1;
01921     int i = (ihi + ilo) >> 1;
01922
01923     if (xx[i] < xx[i + 1])
01924         while (ihi > ilo + 1) {
01925             i = (ihi + ilo) >> 1;
01926             if (xx[i] > x)
01927                 ihi = i;
01928             else
01929                 ilo = i;
01930         } else
01931             while (ihi > ilo + 1) {
01932                 i = (ihi + ilo) >> 1;
01933                 if (xx[i] <= x)
01934                     ihi = i;
01935                 else
01936                     ilo = i;
01937             }
01938
01939     return ilo;
01940 }
```

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

Find array index for regular grid.

Definition at line 1944 of file libtrac.c.

```

01947         {
01948
01949     /* Calculate index... */
01950     int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
01951
01952     /* Check range... */
01953     if (i < 0)
01954         i = 0;
01955     else if (i >= n - 2)
01956         i = n - 2;
01957
01958     return i;
01959 }
```

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

Read atmospheric data.

Definition at line 1963 of file libtrac.c.

```

01966         {
01967
01968     FILE *in;
01969
01970     char line[LEN], *tok;
01971
01972     double t0;
01973
01974     int dimid, ip, iq, ncid, varid;
01975
01976     size_t nparts;
01977
01978     /* Init... */
```

```

01979 atm->np = 0;
01980
01981 /* Write info... */
01982 printf("Read atmospheric data: %s\n", filename);
01983
01984 /* Read ASCII data... */
01985 if (ctl->atm_type == 0) {
01986
01987     /* Open file... */
01988     if (!(in = fopen(filename, "r"))) {
01989         WARN("File not found!");
01990         return 0;
01991     }
01992
01993     /* Read line... */
01994     while (fgets(line, LEN, in)) {
01995
01996         /* Read data... */
01997         TOK(line, tok, "%lg", atm->time[atm->np]);
01998         TOK(NULL, tok, "%lg", atm->p[atm->np]);
01999         TOK(NULL, tok, "%lg", atm->lon[atm->np]);
02000         TOK(NULL, tok, "%lg", atm->lat[atm->np]);
02001         for (iq = 0; iq < ctl->nq; iq++)
02002             TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
02003
02004         /* Convert altitude to pressure... */
02005         atm->p[atm->np] = P(atm->p[atm->np]);
02006
02007         /* Increment data point counter... */
02008         if (++atm->np > NP)
02009             ERRMSG("Too many data points!");
02010     }
02011
02012     /* Close file... */
02013     fclose(in);
02014 }
02015
02016 /* Read binary data... */
02017 else if (ctl->atm_type == 1) {
02018
02019     /* Open file... */
02020     if (!(in = fopen(filename, "r")))
02021         return 0;
02022
02023     /* Read data... */
02024     FREAD(&atm->np, int, 1, in);
02025     FREAD(atm->time, double,
02026           (size_t) atm->np,
02027           in);
02028     FREAD(atm->p, double,
02029           (size_t) atm->np,
02030           in);
02031     FREAD(atm->lon, double,
02032           (size_t) atm->np,
02033           in);
02034     FREAD(atm->lat, double,
02035           (size_t) atm->np,
02036           in);
02037     for (iq = 0; iq < ctl->nq; iq++)
02038         FREAD(atm->q[iq], double,
02039               (size_t) atm->np,
02040               in);
02041
02042     /* Close file... */
02043     fclose(in);
02044 }
02045
02046 /* Read netCDF data... */
02047 else if (ctl->atm_type == 2) {
02048
02049     /* Open file... */
02050     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02051         return 0;
02052
02053     /* Get dimensions... */
02054     NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
02055     NC(nc_inq_dimlen(ncid, dimid, &nparts));
02056     atm->np = (int) nparts;
02057     if (atm->np > NP)
02058         ERRMSG("Too many particles!");
02059
02060     /* Get time... */
02061     NC(nc_inq_varid(ncid, "time", &varid));
02062     NC(nc_get_var_double(ncid, varid, &t0));
02063     for (ip = 0; ip < atm->np; ip++)
02064         atm->time[ip] = t0;
02065

```

```

02066  /* Read geolocations... */
02067  NC(nc_inq_varid(ncid, "PRESS", &varid));
02068  NC(nc_get_var_double(ncid, varid, atm->p));
02069  NC(nc_inq_varid(ncid, "LON", &varid));
02070  NC(nc_get_var_double(ncid, varid, atm->lon));
02071  NC(nc_inq_varid(ncid, "LAT", &varid));
02072  NC(nc_get_var_double(ncid, varid, atm->lat));
02073
02074  /* Read variables... */
02075  if (ctl->qnt_p >= 0)
02076      if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
02077          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
02078  if (ctl->qnt_t >= 0)
02079      if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
02080          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
02081  if (ctl->qnt_u >= 0)
02082      if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
02083          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
02084  if (ctl->qnt_v >= 0)
02085      if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
02086          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
02087  if (ctl->qnt_w >= 0)
02088      if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
02089          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
02090  if (ctl->qnt_h2o >= 0)
02091      if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
02092          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
02093  if (ctl->qnt_o3 >= 0)
02094      if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
02095          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
02096  if (ctl->qnt_theta >= 0)
02097      if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
02098          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
02099  if (ctl->qnt_pv >= 0)
02100      if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
02101          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
02102
02103  /* Check data... */
02104  for (ip = 0; ip < atm->np; ip++)
02105      if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
02106          || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
02107          || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
02108          || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
02109          || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
02110      atm->time[ip] = GSL_NAN;
02111      atm->p[ip] = GSL_NAN;
02112      atm->lon[ip] = GSL_NAN;
02113      atm->lat[ip] = GSL_NAN;
02114      for (iq = 0; iq < ctl->nq; iq++)
02115          atm->q[iq][ip] = GSL_NAN;
02116      } else {
02117          if (ctl->qnt_h2o >= 0)
02118              atm->q[ctl->qnt_h2o][ip] *= 1.608;
02119          if (ctl->qnt_pv >= 0)
02120              atm->q[ctl->qnt_pv][ip] *= 1e6;
02121          if (atm->lon[ip] > 180)
02122              atm->lon[ip] -= 360;
02123      }
02124
02125  /* Close file... */
02126  NC(nc_close(ncid));
02127  }
02128
02129  /* Error... */
02130  else
02131      ERRMSG("Atmospheric data type not supported!");
02132
02133  /* Check number of points... */
02134  if (atm->np < 1)
02135      ERRMSG("Can not read any data!");
02136
02137  /* Return success... */
02138  return 1;
02139  }

```

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

Read control parameters.

Definition at line 2143 of file libtrac.c.


```

02147         {
02148
02149         /* Write info... */
02150         printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
02151             "(executable: %s | compiled: %s, %s)\n\n",
02152             argv[0], __DATE__, __TIME__);
02153
02154         /* Initialize quantity indices... */
02155         ctl->qnt_ens = -1;
02156         ctl->qnt_m = -1;
02157         ctl->qnt_r = -1;
02158         ctl->qnt_rho = -1;
02159         ctl->qnt_ps = -1;
02160         ctl->qnt_pt = -1;
02161         ctl->qnt_z = -1;
02162         ctl->qnt_p = -1;
02163         ctl->qnt_t = -1;
02164         ctl->qnt_u = -1;
02165         ctl->qnt_v = -1;
02166         ctl->qnt_w = -1;
02167         ctl->qnt_h2o = -1;
02168         ctl->qnt_o3 = -1;
02169         ctl->qnt_lwc = -1;
02170         ctl->qnt_iwc = -1;
02171         ctl->qnt_pc = -1;
02172         ctl->qnt_hno3 = -1;
02173         ctl->qnt_oh = -1;
02174         ctl->qnt_rh = -1;
02175         ctl->qnt_theta = -1;
02176         ctl->qnt_vh = -1;
02177         ctl->qnt_vz = -1;
02178         ctl->qnt_pv = -1;
02179         ctl->qnt_tice = -1;
02180         ctl->qnt_tsts = -1;
02181         ctl->qnt_tnat = -1;
02182         ctl->qnt_stat = -1;
02183
02184         /* Read quantities... */
02185         ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
02186         if (ctl->nq > NQ)
02187             ERRMSG("Too many quantities!");
02188         for (int iq = 0; iq < ctl->nq; iq++) {
02189
02190             /* Read quantity name and format... */
02191             scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
02192             scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
02193                 ctl->qnt_format[iq]);
02194
02195             /* Try to identify quantity... */
02196             if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
02197                 ctl->qnt_ens = iq;
02198                 sprintf(ctl->qnt_unit[iq], "-");
02199             } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
02200                 ctl->qnt_m = iq;
02201                 sprintf(ctl->qnt_unit[iq], "kg");
02202             } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
02203                 ctl->qnt_r = iq;
02204                 sprintf(ctl->qnt_unit[iq], "m");
02205             } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
02206                 ctl->qnt_rho = iq;
02207                 sprintf(ctl->qnt_unit[iq], "kg/m^3");
02208             } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
02209                 ctl->qnt_ps = iq;
02210                 sprintf(ctl->qnt_unit[iq], "hPa");
02211             } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
02212                 ctl->qnt_pt = iq;
02213                 sprintf(ctl->qnt_unit[iq], "hPa");
02214             } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
02215                 ctl->qnt_z = iq;
02216                 sprintf(ctl->qnt_unit[iq], "km");
02217             } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
02218                 ctl->qnt_p = iq;
02219                 sprintf(ctl->qnt_unit[iq], "hPa");
02220             } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
02221                 ctl->qnt_t = iq;
02222                 sprintf(ctl->qnt_unit[iq], "K");
02223             } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
02224                 ctl->qnt_u = iq;
02225                 sprintf(ctl->qnt_unit[iq], "m/s");
02226             } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
02227                 ctl->qnt_v = iq;
02228                 sprintf(ctl->qnt_unit[iq], "m/s");
02229             } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
02230                 ctl->qnt_w = iq;
02231                 sprintf(ctl->qnt_unit[iq], "hPa/s");
02232             } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
02233                 ctl->qnt_h2o = iq;

```

```

02234     sprintf(ctl->qnt_unit[iq], "ppv");
02235 } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
02236     ctl->qnt_o3 = iq;
02237     sprintf(ctl->qnt_unit[iq], "ppv");
02238 } else if (strcmp(ctl->qnt_name[iq], "lwc") == 0) {
02239     ctl->qnt_lwc = iq;
02240     sprintf(ctl->qnt_unit[iq], "kg/kg");
02241 } else if (strcmp(ctl->qnt_name[iq], "iwc") == 0) {
02242     ctl->qnt_iwc = iq;
02243     sprintf(ctl->qnt_unit[iq], "kg/kg");
02244 } else if (strcmp(ctl->qnt_name[iq], "pc") == 0) {
02245     ctl->qnt_pc = iq;
02246     sprintf(ctl->qnt_unit[iq], "hPa");
02247 } else if (strcmp(ctl->qnt_name[iq], "hno3") == 0) {
02248     ctl->qnt_hno3 = iq;
02249     sprintf(ctl->qnt_unit[iq], "ppv");
02250 } else if (strcmp(ctl->qnt_name[iq], "oh") == 0) {
02251     ctl->qnt_oh = iq;
02252     sprintf(ctl->qnt_unit[iq], "molec/cm^3");
02253 } else if (strcmp(ctl->qnt_name[iq], "rh") == 0) {
02254     ctl->qnt_rh = iq;
02255     sprintf(ctl->qnt_unit[iq], "%");
02256 } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
02257     ctl->qnt_theta = iq;
02258     sprintf(ctl->qnt_unit[iq], "K");
02259 } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
02260     ctl->qnt_vh = iq;
02261     sprintf(ctl->qnt_unit[iq], "m/s");
02262 } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
02263     ctl->qnt_vz = iq;
02264     sprintf(ctl->qnt_unit[iq], "m/s");
02265 } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
02266     ctl->qnt_pv = iq;
02267     sprintf(ctl->qnt_unit[iq], "PVU");
02268 } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
02269     ctl->qnt_tice = iq;
02270     sprintf(ctl->qnt_unit[iq], "K");
02271 } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
02272     ctl->qnt_tsts = iq;
02273     sprintf(ctl->qnt_unit[iq], "K");
02274 } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
02275     ctl->qnt_tnat = iq;
02276     sprintf(ctl->qnt_unit[iq], "K");
02277 } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
02278     ctl->qnt_stat = iq;
02279     sprintf(ctl->qnt_unit[iq], "-");
02280 } else
02281     scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
02282 }
02283
02284 /* Time steps of simulation... */
02285 ctl->direction =
02286     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
02287 if (ctl->direction != -1 && ctl->direction != 1)
02288     ERRMSG("Set DIRECTION to -1 or 1!");
02289 ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
02290 ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
02291
02292 /* Meteorological data... */
02293 ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
02294 ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
02295 ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
02296 ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
02297 ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
02298 ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
02299 ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
02300 ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
02301 if (ctl->met_np > EP)
02302     ERRMSG("Too many levels!");
02303 for (int ip = 0; ip < ctl->met_np; ip++)
02304     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
02305 ctl->met_tropo =
02306     (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
02307 scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
02308 ctl->met_dt_out =
02309     scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
02310
02311 /* Isosurface parameters... */
02312 ctl->isosurf =
02313     (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
02314 scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
02315
02316 /* Diffusion parameters... */
02317 ctl->turb_dx_trop =
02318     scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
02319 ctl->turb_dx_strat =
02320     scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);

```

```

02321     ctl->turb_dz_trop =
02322         scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
02323     ctl->turb_dz_strat =
02324         scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
02325     ctl->turb_mesox =
02326         scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
02327     ctl->turb_mesoz =
02328         scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
02329
02330     /* Species parameters... */
02331     scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
02332     if (strcmp(ctl->species, "SO2") == 0) {
02333         ctl->molmass = 64.066;
02334         ctl->oh_chem[0] = 3.3e-31; /* (JPL Publication 15-10) */
02335         ctl->oh_chem[1] = 4.3; /* (JPL Publication 15-10) */
02336         ctl->oh_chem[2] = 1.6e-12; /* (JPL Publication 15-10) */
02337         ctl->oh_chem[3] = 0.0; /* (JPL Publication 15-10) */
02338         ctl->wet_depo[0] = 2.0e-05; /* (FLEXPART v10.4) */
02339         ctl->wet_depo[1] = 0.62; /* (FLEXPART v10.4) */
02340         ctl->wet_depo[2] = 1.3e-2; /* (Sander, 2015) */
02341         ctl->wet_depo[3] = 2900.0; /* (Sander, 2015) */
02342     } else {
02343         ctl->molmass =
02344             scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
02345         ctl->tdec_trop =
02346             scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
02347         ctl->tdec_strat =
02348             scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
02349         for (int ip = 0; ip < 4; ip++)
02350             ctl->oh_chem[ip] =
02351                 scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
02352         for (int ip = 0; ip < 4; ip++)
02353             ctl->wet_depo[ip] =
02354                 scan_ctl(filename, argc, argv, "WET_DEPO", ip, "0", NULL);
02355     }
02356
02357     /* PSC analysis... */
02358     ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
02359     ctl->psc_hno3 =
02360         scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
02361
02362     /* Output of atmospheric data... */
02363     scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
atm_basename);
02364     scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
02365     ctl->atm_dt_out =
02366         scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
02367     ctl->atm_filter =
02368         (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
02369     ctl->atm_stride =
02370         (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
02371     ctl->atm_type =
02372         (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
02373
02374     /* Output of CSI data... */
02375     scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
csi_basename);
02376     ctl->csi_dt_out =
02377         scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
02378     scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->
csi_obsfile);
02379     ctl->csi_obsmin =
02380         scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
02381     ctl->csi_modmin =
02382         scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
02383     ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
02384     ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
02385     ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
02386     ctl->csi_lon0 =
02387         scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
02388     ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
02389     ctl->csi_nx =
02390         (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
02391     ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
02392     ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
02393     ctl->csi_ny =
02394         (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
02395
02396     /* Output of ensemble data... */
02397     scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
ens_basename);
02398
02399     /* Output of grid data... */
02400     scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
02401         ctl->grid_basename);
02402     scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
grid_gpfile);

```

```

02403     ctl->grid_dt_out =
02404         scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
02405     ctl->grid_sparse =
02406         (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
02407     ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
02408     ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
02409     ctl->grid_nz =
02410         (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
02411     ctl->grid_lon0 =
02412         scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
02413     ctl->grid_lon1 =
02414         scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
02415     ctl->grid_nx =
02416         (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
02417     ctl->grid_lat0 =
02418         scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
02419     ctl->grid_lat1 =
02420         scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
02421     ctl->grid_ny =
02422         (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
02423
02424     /* Output of profile data... */
02425     scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
02426         ctl->prof_basename);
02427     scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
02428     prof_obsfile);
02429     ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
02430     ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
02431     ctl->prof_nz =
02432         (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
02433     ctl->prof_lon0 =
02434         scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
02435     ctl->prof_lon1 =
02436         scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
02437     ctl->prof_nx =
02438         (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
02439     ctl->prof_lat0 =
02440         scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
02441     ctl->prof_lat1 =
02442         scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
02443     ctl->prof_ny =
02444         (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
02445
02446     /* Output of station data... */
02447     scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
02448         ctl->stat_basename);
02449     ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
02450     ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
02451     ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
02452 }

```

Here is the call graph for this function:



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

Read meteorological data file.

Definition at line 2455 of file libtrac.c.

```

02458     {
02459
02460     char cmd[2 * LEN], levname[LEN], tstr[10];

```

```

02461
02462     int ip, dimid, ncid, varid, year, mon, day, hour;
02463
02464     size_t np, nx, ny;
02465
02466     /* Write info... */
02467     printf("Read meteorological data: %s\n", filename);
02468
02469     /* Get time from filename... */
02470     sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
02471     year = atoi(tstr);
02472     sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
02473     mon = atoi(tstr);
02474     sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
02475     day = atoi(tstr);
02476     sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
02477     hour = atoi(tstr);
02478     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
02479
02480     /* Open netCDF file... */
02481     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
02482
02483         /* Try to stage meteo file... */
02484         if (ctl->met_stage[0] != '-') {
02485             sprintf(cmd, "%s %d %02d %02d %s", ctl->met_stage,
02486                     year, mon, day, hour, filename);
02487             if (system(cmd) != 0)
02488                 ERRMSG("Error while staging meteo data!");
02489         }
02490
02491         /* Try to open again... */
02492         if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
02493             WARN("File not found!");
02494             return 0;
02495         }
02496     }
02497
02498     /* Get dimensions... */
02499     NC(nc_inq_dimid(ncid, "lon", &dimid));
02500     NC(nc_inq_dimlen(ncid, dimid, &nx));
02501     if (nx < 2 || nx > EX)
02502         ERRMSG("Number of longitudes out of range!");
02503
02504     NC(nc_inq_dimid(ncid, "lat", &dimid));
02505     NC(nc_inq_dimlen(ncid, dimid, &ny));
02506     if (ny < 2 || ny > EY)
02507         ERRMSG("Number of latitudes out of range!");
02508
02509     sprintf(levname, "lev");
02510     NC(nc_inq_dimid(ncid, levname, &dimid));
02511     NC(nc_inq_dimlen(ncid, dimid, &np));
02512     if (np == 1) {
02513         sprintf(levname, "lev_2");
02514         if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
02515             sprintf(levname, "plev");
02516             nc_inq_dimid(ncid, levname, &dimid);
02517         }
02518         NC(nc_inq_dimlen(ncid, dimid, &np));
02519     }
02520     if (np < 2 || np > EP)
02521         ERRMSG("Number of levels out of range!");
02522
02523     /* Store dimensions... */
02524     met->np = (int) np;
02525     met->nx = (int) nx;
02526     met->ny = (int) ny;
02527
02528     /* Get horizontal grid... */
02529     NC(nc_inq_varid(ncid, "lon", &varid));
02530     NC(nc_get_var_double(ncid, varid, met->lon));
02531     NC(nc_inq_varid(ncid, "lat", &varid));
02532     NC(nc_get_var_double(ncid, varid, met->lat));
02533
02534     /* Read meteorological data... */
02535     if (!read_met_help_3d(ncid, "t", "T", met, met->t, 1.0))
02536         ERRMSG("Cannot read temperature!");
02537     if (!read_met_help_3d(ncid, "u", "U", met, met->u, 1.0))
02538         ERRMSG("Cannot read zonal wind!");
02539     if (!read_met_help_3d(ncid, "v", "V", met, met->v, 1.0))
02540         ERRMSG("Cannot read meridional wind!");
02541     if (!read_met_help_3d(ncid, "w", "W", met, met->w, 0.01f))
02542         WARN("Cannot read vertical velocity!");
02543     if (!read_met_help_3d(ncid, "q", "Q", met, met->h2o, (float) (MA / MH2O)))
02544         WARN("Cannot read specific humidity!");
02545     if (!read_met_help_3d(ncid, "o3", "O3", met, met->o3, (float) (MA / MO3)))
02546         WARN("Cannot read ozone data!");
02547     if (!read_met_help_3d(ncid, "clwc", "CLWC", met, met->lwc, 1.0))

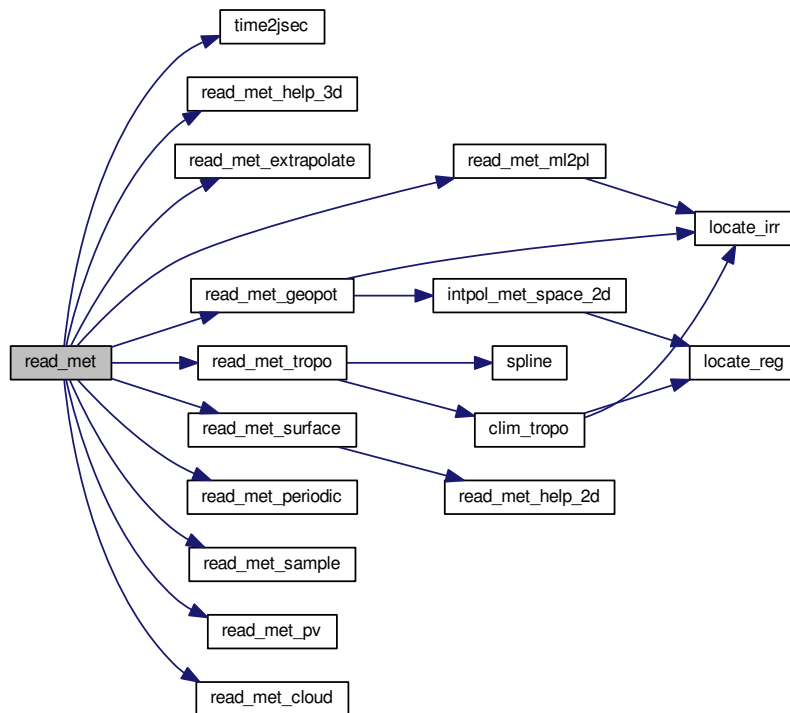
```

```

02548     WARN("Cannot read cloud liquid water content!");
02549     if (!read_met_help_3d(ncid, "ciwc", "CIWC", met, met->iwc, 1.0))
02550         WARN("Cannot read cloud ice water content!");
02551
02552     /* Meteo data on pressure levels... */
02553     if (ctl->met_np <= 0) {
02554
02555         /* Read pressure levels from file... */
02556         NC(nc_inq_varid(ncid, levname, &varid));
02557         NC(nc_get_var_double(ncid, varid, met->p));
02558         for (ip = 0; ip < met->np; ip++)
02559             met->p[ip] /= 100.;
02560
02561         /* Extrapolate data for lower boundary... */
02562         read_met_extrapolate(met);
02563     }
02564
02565     /* Meteo data on model levels... */
02566     else {
02567
02568         /* Read pressure data from file... */
02569         read_met_help_3d(ncid, "pl", "PL", met, met->pl, 0.01f);
02570
02571         /* Interpolate from model levels to pressure levels... */
02572         read_met_ml2pl(ctl, met, met->t);
02573         read_met_ml2pl(ctl, met, met->u);
02574         read_met_ml2pl(ctl, met, met->v);
02575         read_met_ml2pl(ctl, met, met->w);
02576         read_met_ml2pl(ctl, met, met->h2o);
02577         read_met_ml2pl(ctl, met, met->o3);
02578         read_met_ml2pl(ctl, met, met->lwc);
02579         read_met_ml2pl(ctl, met, met->iwc);
02580
02581         /* Set pressure levels... */
02582         met->np = ctl->met_np;
02583         for (ip = 0; ip < met->np; ip++)
02584             met->p[ip] = ctl->met_p[ip];
02585     }
02586
02587     /* Check ordering of pressure levels... */
02588     for (ip = 1; ip < met->np; ip++)
02589         if (met->p[ip - 1] < met->p[ip])
02590             ERRMSG("Pressure levels must be descending!");
02591
02592     /* Read surface data... */
02593     read_met_surface(ncid, met);
02594
02595     /* Create periodic boundary conditions... */
02596     read_met_periodic(met);
02597
02598     /* Downsampling... */
02599     read_met_sample(ctl, met);
02600
02601     /* Calculate geopotential heights... */
02602     read_met_geopot(met);
02603
02604     /* Calculate potential vorticity... */
02605     read_met_pv(met);
02606
02607     /* Calculate tropopause pressure... */
02608     read_met_tropo(ctl, met);
02609
02610     /* Calculate cloud properties... */
02611     read_met_cloud(met);
02612
02613     /* Close file... */
02614     NC(nc_close(ncid));
02615
02616     /* Return success... */
02617     return 1;
02618 }

```

Here is the call graph for this function:



5.19.2.21 void read_met_cloud (met_t * met)

Calculate cloud properties.

Definition at line 2622 of file libtrac.c.

```

02623     {
02624
02625     int ix, iy, ip;
02626
02627     /* Loop over columns... */
02628     #pragma omp parallel for default(shared) private(ix,iy,ip)
02629     for (ix = 0; ix < met->nx; ix++)
02630         for (iy = 0; iy < met->ny; iy++) {
02631
02632         /* Init... */
02633         met->pc[ix][iy] = GSL_NAN;
02634         met->cl[ix][iy] = 0;
02635
02636         /* Loop over pressure levels... */
02637         for (ip = 0; ip < met->np - 1; ip++) {
02638
02639         /* Check pressure... */
02640         if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
02641             continue;
02642
02643         /* Get cloud top pressure ... */
02644         if (met->iwc[ix][iy][ip] > 0 || met->lwc[ix][iy][ip] > 0)
02645             met->pc[ix][iy] = (float) met->p[ip + 1];
02646
02647         /* Get cloud water... */
02648         met->cl[ix][iy] += (float)
02649             (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1]
02650                 + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
02651              * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
02652         }
02653     }
02654 }

```

5.19.2.22 void read_met_extrapolate (met_t * met)

Extrapolate meteorological data at lower boundary.

Definition at line 2658 of file libtrac.c.

```

02659         {
02660
02661     int ip, ip0, ix, iy;
02662
02663     /* Loop over columns... */
02664     #pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
02665     for (ix = 0; ix < met->nx; ix++)
02666         for (iy = 0; iy < met->ny; iy++) {
02667
02668         /* Find lowest valid data point... */
02669         for (ip0 = met->np - 1; ip0 >= 0; ip0--)
02670             if (!isfinite(met->t[ix][iy][ip0])
02671                 || !isfinite(met->u[ix][iy][ip0])
02672                 || !isfinite(met->v[ix][iy][ip0])
02673                 || !isfinite(met->w[ix][iy][ip0]))
02674             break;
02675
02676         /* Extrapolate... */
02677         for (ip = ip0; ip >= 0; ip--) {
02678             met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
02679             met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
02680             met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
02681             met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
02682             met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
02683             met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
02684             met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
02685             met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
02686         }
02687     }
02688 }
```

5.19.2.23 void read_met_geopot (met_t * met)

Calculate geopotential heights.

Definition at line 2692 of file libtrac.c.

```

02693         {
02694
02695     const int dx = 6, dy = 4;
02696
02697     static float help[EX][EY][EP];
02698
02699     double logp[EP], ts, z0, cw[3];
02700
02701     int ip, ip0, ix, ix2, ix3, iy, iy2, n, ci[3];
02702
02703     /* Calculate log pressure... */
02704     for (ip = 0; ip < met->np; ip++)
02705         logp[ip] = log(met->p[ip]);
02706
02707     /* Initialize geopotential heights... */
02708     #pragma omp parallel for default(shared) private(ix,iy,ip)
02709     for (ix = 0; ix < met->nx; ix++)
02710         for (iy = 0; iy < met->ny; iy++)
02711             for (ip = 0; ip < met->np; ip++)
02712                 met->z[ix][iy][ip] = GSL_NAN;
02713
02714     /* Apply hydrostatic equation to calculate geopotential heights... */
02715     #pragma omp parallel for default(shared) private(ix,iy,z0,ip0,ts,ip,ci,cw)
02716     for (ix = 0; ix < met->nx; ix++)
02717         for (iy = 0; iy < met->ny; iy++) {
02718
02719         /* Get surface height... */
02720         intpol_met_space_2d(met, met->z0, met->lon[ix], met->
02721 lat[iy], &z0, ci,
02722 cw, 1);
02723
02724         /* Find surface pressure level index... */
02725         ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
```

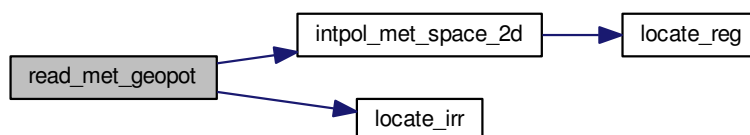


```

02725
02726 /* Get virtual temperature at the surface... */
02727 ts =
02728     LIN(met->p[ip0],
02729         TVIRT(met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]),
02730         met->p[ip0 + 1],
02731         TVIRT(met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]),
02732         met->ps[ix][iy]);
02733
02734 /* Upper part of profile... */
02735 met->z[ix][iy][ip0 + 1]
02736     = (float) (z0 + RI / MA / G0 * 0.5
02737               * (ts + TVIRT(met->t[ix][iy][ip0 + 1],
02738                             met->h2o[ix][iy][ip0 + 1]))
02739               * (log(met->ps[ix][iy]) - logp[ip0 + 1]));
02740 for (ip = ip0 + 2; ip < met->np; ip++)
02741     met->z[ix][iy][ip]
02742         = (float) (met->z[ix][iy][ip - 1] + RI / MA / G0 * 0.5 *
02743                 (TVIRT(met->t[ix][iy][ip - 1], met->h2o[ix][iy][ip - 1])
02744                  + TVIRT(met->t[ix][iy][ip], met->h2o[ix][iy][ip]))
02745                 * (logp[ip - 1] - logp[ip]));
02746 }
02747
02748 /* Horizontal smoothing... */
02749 #pragma omp parallel for default(shared) private(ix,iy,ip,n,ix2,ix3,iy2)
02750 for (ix = 0; ix < met->nx; ix++)
02751     for (iy = 0; iy < met->ny; iy++)
02752         for (ip = 0; ip < met->np; ip++) {
02753             n = 0;
02754             help[ix][iy][ip] = 0;
02755             for (ix2 = ix - dx; ix2 <= ix + dx; ix2++) {
02756                 ix3 = ix2;
02757                 if (ix3 < 0)
02758                     ix3 += met->nx;
02759                 else if (ix3 >= met->nx)
02760                     ix3 -= met->nx;
02761                 for (iy2 = GSL_MAX(iy - dy, 0);
02762                     iy2 <= GSL_MIN(iy + dy, met->ny - 1); iy2++)
02763                     if (isfinite(met->z[ix3][iy2][ip])) {
02764                         help[ix][iy][ip] += met->z[ix3][iy2][ip];
02765                         n++;
02766                     }
02767             }
02768             if (n > 0)
02769                 help[ix][iy][ip] /= (float) n;
02770             else
02771                 help[ix][iy][ip] = GSL_NAN;
02772         }
02773
02774 /* Copy data... */
02775 #pragma omp parallel for default(shared) private(ix,iy,ip)
02776 for (ix = 0; ix < met->nx; ix++)
02777     for (iy = 0; iy < met->ny; iy++)
02778         for (ip = 0; ip < met->np; ip++)
02779             met->z[ix][iy][ip] = help[ix][iy][ip];
02780 }

```

Here is the call graph for this function:



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

Read and convert 3D variable from meteorological data file.

Definition at line 2784 of file libtrac.c.

```

02790         {
02791
02792     float *help;
02793
02794     int ip, ix, iy, varid;
02795
02796     /* Check if variable exists... */
02797     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
02798         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
02799             return 0;
02800
02801     /* Allocate... */
02802     ALLOC(help, float, EX * EY * EP);
02803
02804     /* Read data... */
02805     NC(nc_get_var_float(ncid, varid, help));
02806
02807     /* Copy and check data... */
02808 #pragma omp parallel for default(shared) private(ix,iy,ip)
02809     for (ix = 0; ix < met->nx; ix++)
02810         for (iy = 0; iy < met->ny; iy++)
02811             for (ip = 0; ip < met->np; ip++) {
02812                 dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
02813                 if (fabsf(dest[ix][iy][ip]) < 1e14f)
02814                     dest[ix][iy][ip] *= scl;
02815                 else
02816                     dest[ix][iy][ip] = GSL_NAN;
02817             }
02818
02819     /* Free... */
02820     free(help);
02821
02822     /* Return... */
02823     return 1;
02824 }

```

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

Read and convert 2D variable from meteorological data file.

Definition at line [2828](#) of file [libtrac.c](#).

```

02834         {
02835
02836     float *help;
02837
02838     int ix, iy, varid;
02839
02840     /* Check if variable exists... */
02841     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
02842         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
02843             return 0;
02844
02845     /* Allocate... */
02846     ALLOC(help, float, EX * EY);
02847
02848     /* Read data... */
02849     NC(nc_get_var_float(ncid, varid, help));
02850
02851     /* Copy and check data... */
02852 #pragma omp parallel for default(shared) private(ix,iy)
02853     for (ix = 0; ix < met->nx; ix++)
02854         for (iy = 0; iy < met->ny; iy++) {
02855             dest[ix][iy] = help[iy * met->nx + ix];
02856             if (fabsf(dest[ix][iy]) < 1e14f)
02857                 dest[ix][iy] *= scl;
02858             else
02859                 dest[ix][iy] = GSL_NAN;
02860         }
02861
02862     /* Free... */
02863     free(help);
02864
02865     /* Return... */
02866     return 1;
02867 }

```

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

Convert meteorological data from model levels to pressure levels.

Definition at line 2871 of file [libtrac.c](#).

```

02874         {
02875
02876     double aux[EP], p[EP], pt;
02877
02878     int ip, ip2, ix, iy;
02879
02880     /* Loop over columns... */
02881     #pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
02882     for (ix = 0; ix < met->nx; ix++)
02883         for (iy = 0; iy < met->ny; iy++) {
02884
02885         /* Copy pressure profile... */
02886         for (ip = 0; ip < met->np; ip++)
02887             p[ip] = met->pl[ix][iy][ip];
02888
02889         /* Interpolate... */
02890         for (ip = 0; ip < ctl->met_np; ip++) {
02891             pt = ctl->met_p[ip];
02892             if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
02893                 pt = p[0];
02894             else if ((pt > p[met->np - 1] && p[1] > p[0])
02895                     || (pt < p[met->np - 1] && p[1] < p[0]))
02896                 pt = p[met->np - 1];
02897             ip2 = locate_irr(p, met->np, pt);
02898             aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
02899                         p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
02900         }
02901
02902         /* Copy data... */
02903         for (ip = 0; ip < ctl->met_np; ip++)
02904             var[ix][iy][ip] = (float) aux[ip];
02905     }
02906 }
```

Here is the call graph for this function:

5.19.2.27 void read_met_periodic (met_t * *met*)

Create meteorological data with periodic boundary conditions.

Definition at line 2910 of file [libtrac.c](#).

```

02911         {
02912
02913     /* Check longitudes... */
02914     if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
02915               + met->lon[1] - met->lon[0] - 360) < 0.01))
02916         return;
02917
02918     /* Increase longitude counter... */
02919     if ((++met->nx) > EX)
02920         ERRMSG("Cannot create periodic boundary conditions!");
```

```

02921
02922  /* Set longitude... */
02923  met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
lon[0];
02924
02925  /* Loop over latitudes and pressure levels... */
02926  #pragma omp parallel for default(shared)
02927  for (int iy = 0; iy < met->ny; iy++) {
02928      met->ps[met->nx - 1][iy] = met->ps[0][iy];
02929      met->zs[met->nx - 1][iy] = met->zs[0][iy];
02930      for (int ip = 0; ip < met->np; ip++) {
02931          met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
02932          met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
02933          met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
02934          met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
02935          met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
02936          met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
02937          met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
02938          met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
02939      }
02940  }
02941 }

```

5.19.2.28 void read_met_pv (met_t * met)

Calculate potential vorticity.

Definition at line 2945 of file libtrac.c.

```

02946      {
02947
02948      double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
02949      dtdp, dudp, dvdp, latr, vort, pows[EP];
02950
02951      int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
02952
02953      /* Set powers... */
02954      for (ip = 0; ip < met->np; ip++)
02955          pows[ip] = pow(1000. / met->p[ip], 0.286);
02956
02957      /* Loop over grid points... */
02958      #pragma omp parallel for default(shared)
02959      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)
02960      for (ix = 0; ix < met->nx; ix++) {
02961
02962          /* Set indices... */
02963          ix0 = GSL_MAX(ix - 1, 0);
02964          ix1 = GSL_MIN(ix + 1, met->nx - 1);
02965
02966          /* Loop over grid points... */
02967          for (iy = 0; iy < met->ny; iy++) {
02968
02969              /* Set indices... */
02970              iy0 = GSL_MAX(iy - 1, 0);
02971              iy1 = GSL_MIN(iy + 1, met->ny - 1);
02972
02973              /* Set auxiliary variables... */
02974              latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
02975              dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
02976              dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
02977              c0 = cos(met->lat[iy0] / 180. * M_PI);
02978              c1 = cos(met->lat[iy1] / 180. * M_PI);
02979              cr = cos(latr / 180. * M_PI);
02980              vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
02981
02982              /* Loop over grid points... */
02983              for (ip = 0; ip < met->np; ip++) {
02984
02985                  /* Get gradients in longitude... */
02986                  dtdx = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
02987                  dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
02988
02989                  /* Get gradients in latitude... */
02990                  dtdy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
02991                  dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
02992
02993                  /* Set indices... */
02994                  ip0 = GSL_MAX(ip - 1, 0);
02995                  ip1 = GSL_MIN(ip + 1, met->np - 1);
02996
02997              }
02998          }
02999      }

```

```

02996      /* Get gradients in pressure... */
02997      dp0 = 100. * (met->p[ip] - met->p[ip0]);
02998      dp1 = 100. * (met->p[ip1] - met->p[ip]);
02999      if (ip != ip0 && ip != ip1) {
03000          denom = dp0 * dp1 * (dp0 + dp1);
03001          dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
03002                - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
03003                + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
03004              / denom;
03005          dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
03006                - dp1 * dp1 * met->u[ix][iy][ip0]
03007                + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
03008              / denom;
03009          dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
03010                - dp1 * dp1 * met->v[ix][iy][ip0]
03011                + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
03012              / denom;
03013      } else {
03014          denom = dp0 + dp1;
03015          dtdp =
03016              (met->t[ix][iy][ip1] * pows[ip1] -
03017               met->t[ix][iy][ip0] * pows[ip0]) / denom;
03018          dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
03019          dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
03020      }
03021
03022      /* Calculate PV... */
03023      met->pv[ix][iy][ip] = (float)
03024          (1e6 * G0 *
03025           (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
03026  }
03027  }
03028  }
03029
03030  /* Fix for polar regions... */
03031  #pragma omp parallel for default(shared) private(ix,ip)
03032  for (ix = 0; ix < met->nx; ix++)
03033      for (ip = 0; ip < met->np; ip++) {
03034          met->pv[ix][0][ip]
03035              = met->pv[ix][1][ip]
03036              = met->pv[ix][2][ip];
03037          met->pv[ix][met->ny - 1][ip]
03038              = met->pv[ix][met->ny - 2][ip]
03039              = met->pv[ix][met->ny - 3][ip];
03040      }
03041  }

```

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

Downsampling of meteorological data.

Definition at line 3045 of file libtrac.c.

```

03047      {
03048
03049      met_t *help;
03050
03051      float w, wsum;
03052
03053      int ip, ip2, ix, ix2, ix3, iy, iy2;
03054
03055      /* Check parameters... */
03056      if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
03057          && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
03058          return;
03059
03060      /* Allocate... */
03061      ALLOC(help, met_t, 1);
03062
03063      /* Copy data... */
03064      help->nx = met->nx;
03065      help->ny = met->ny;
03066      help->np = met->np;
03067      memcpy(help->lon, met->lon, sizeof(met->lon));
03068      memcpy(help->lat, met->lat, sizeof(met->lat));
03069      memcpy(help->p, met->p, sizeof(met->p));
03070
03071      /* Smoothing... */
03072      for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
03073          for (iy = 0; iy < met->ny; iy += ctl->met_dy) {

```

```

03074     for (ip = 0; ip < met->np; ip += ctl->met_dp) {
03075         help->ps[ix][iy] = 0;
03076         help->zs[ix][iy] = 0;
03077         help->t[ix][iy][ip] = 0;
03078         help->u[ix][iy][ip] = 0;
03079         help->v[ix][iy][ip] = 0;
03080         help->w[ix][iy][ip] = 0;
03081         help->h2o[ix][iy][ip] = 0;
03082         help->o3[ix][iy][ip] = 0;
03083         help->lwc[ix][iy][ip] = 0;
03084         help->iwc[ix][iy][ip] = 0;
03085         wsum = 0;
03086         for (ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1; ix2++) {
03087             ix3 = ix2;
03088             if (ix3 < 0)
03089                 ix3 += met->nx;
03090             else if (ix3 >= met->nx)
03091                 ix3 -= met->nx;
03092
03093             for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
03094                  iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
03095                 for (ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
03096                      ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
03097                     w = (float) (1.0 - abs(ix - ix2) / ctl->met_sx)
03098                         * (float) (1.0 - abs(iy - iy2) / ctl->met_sy)
03099                         * (float) (1.0 - abs(ip - ip2) / ctl->met_sp);
03100                     help->ps[ix][iy] += w * met->ps[ix3][iy2];
03101                     help->zs[ix][iy] += w * met->zs[ix3][iy2];
03102                     help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
03103                     help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
03104                     help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
03105                     help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
03106                     help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
03107                     help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
03108                     help->lwc[ix][iy][ip] += w * met->lwc[ix3][iy2][ip2];
03109                     help->iwc[ix][iy][ip] += w * met->iwc[ix3][iy2][ip2];
03110                     wsum += w;
03111                 }
03112             }
03113         help->ps[ix][iy] /= wsum;
03114         help->zs[ix][iy] /= wsum;
03115         help->t[ix][iy][ip] /= wsum;
03116         help->u[ix][iy][ip] /= wsum;
03117         help->v[ix][iy][ip] /= wsum;
03118         help->w[ix][iy][ip] /= wsum;
03119         help->h2o[ix][iy][ip] /= wsum;
03120         help->o3[ix][iy][ip] /= wsum;
03121         help->lwc[ix][iy][ip] /= wsum;
03122         help->iwc[ix][iy][ip] /= wsum;
03123     }
03124 }
03125 }
03126
03127 /* Downsampling... */
03128 met->nx = 0;
03129 for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
03130     met->lon[met->nx] = help->lon[ix];
03131     met->ny = 0;
03132     for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
03133         met->lat[met->ny] = help->lat[iy];
03134         met->ps[met->nx][met->ny] = help->ps[ix][iy];
03135         met->zs[met->nx][met->ny] = help->zs[ix][iy];
03136         met->np = 0;
03137         for (ip = 0; ip < help->np; ip += ctl->met_dp) {
03138             met->p[met->np] = help->p[ip];
03139             met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
03140             met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
03141             met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
03142             met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
03143             met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
03144             met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
03145             met->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];
03146             met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
03147             met->np++;
03148         }
03149         met->ny++;
03150     }
03151     met->nx++;
03152 }
03153
03154 /* Free... */
03155 free(help);
03156 }

```

5.19.2.30 void read_met_surface (int ncid, met_t * met)

Read surface data.

Definition at line 3160 of file libtrac.c.

```

03162         {
03163
03164     int ix, iy;
03165
03166     /* Read surface pressure... */
03167     if (!read_met_help_2d(ncid, "ps", "PS", met, met->ps, 0.01f)) {
03168         if (!read_met_help_2d(ncid, "lnsp", "LNSP", met, met->ps, 1.0)) {
03169             ERRMSG("Cannot not read surface pressure data!");
03170             for (ix = 0; ix < met->nx; ix++)
03171                 for (iy = 0; iy < met->ny; iy++)
03172                     met->ps[ix][iy] = (float) met->p[0];
03173         } else {
03174             for (iy = 0; iy < met->ny; iy++)
03175                 for (ix = 0; ix < met->nx; ix++)
03176                     met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
03177         }
03178     }
03179
03180     /* Read geopotential height at the surface... */
03181     if (!read_met_help_2d
03182         (ncid, "z", "Z", met, met->zs, (float) (1. / (1000. * G0))))
03183         if (!read_met_help_2d
03184             (ncid, "zm", "ZM", met, met->zs, (float) (1. / 1000.)))
03185             ERRMSG("Cannot read surface geopotential height!");
03186 }

```

Here is the call graph for this function:



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

Calculate tropopause pressure.

Definition at line 3190 of file libtrac.c.

```

03192         {
03193
03194     double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
03195         th2[200], z[EP], z2[200];
03196
03197     int found, ix, iy, iz, iz2;
03198
03199     /* Get altitude and pressure profiles... */
03200     for (iz = 0; iz < met->np; iz++)
03201         z[iz] = Z(met->p[iz]);
03202     for (iz = 0; iz <= 190; iz++) {
03203         z2[iz] = 4.5 + 0.1 * iz;
03204         p2[iz] = P(z2[iz]);
03205     }
03206
03207     /* Do not calculate tropopause... */
03208     if (ctl->met_tropo == 0)
03209         for (ix = 0; ix < met->nx; ix++)

```

```

03210     for (iy = 0; iy < met->ny; iy++)
03211         met->pt[ix][iy] = GSL_NAN;
03212
03213     /* Use tropopause climatology... */
03214     else if (ctl->met_tropo == 1) {
03215 #pragma omp parallel for default(shared) private(ix,iy)
03216         for (ix = 0; ix < met->nx; ix++)
03217             for (iy = 0; iy < met->ny; iy++)
03218                 met->pt[ix][iy] = (float) clim_tropo(met->time, met->lat[iy]);
03219     }
03220
03221     /* Use cold point... */
03222     else if (ctl->met_tropo == 2) {
03223
03224         /* Loop over grid points... */
03225 #pragma omp parallel for default(shared) private(ix,iy,iz,t,t2)
03226         for (ix = 0; ix < met->nx; ix++)
03227             for (iy = 0; iy < met->ny; iy++) {
03228
03229                 /* Interpolate temperature profile... */
03230                 for (iz = 0; iz < met->np; iz++)
03231                     t[iz] = met->t[ix][iy][iz];
03232                 spline(z, t, met->np, z2, t2, 171);
03233
03234                 /* Find minimum... */
03235                 iz = (int) gsl_stats_min_index(t2, 1, 171);
03236                 if (iz > 0 && iz < 170)
03237                     met->pt[ix][iy] = (float) p2[iz];
03238                 else
03239                     met->pt[ix][iy] = GSL_NAN;
03240             }
03241     }
03242
03243     /* Use WMO definition... */
03244     else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
03245
03246         /* Loop over grid points... */
03247 #pragma omp parallel for default(shared) private(ix,iy,iz,iz2,t,t2,found)
03248         for (ix = 0; ix < met->nx; ix++)
03249             for (iy = 0; iy < met->ny; iy++) {
03250
03251                 /* Interpolate temperature profile... */
03252                 for (iz = 0; iz < met->np; iz++)
03253                     t[iz] = met->t[ix][iy][iz];
03254                 spline(z, t, met->np, z2, t2, 191);
03255
03256                 /* Find 1st tropopause... */
03257                 met->pt[ix][iy] = GSL_NAN;
03258                 for (iz = 0; iz <= 170; iz++) {
03259                     found = 1;
03260                     for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
03261                         if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
03262                             * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
03263                             found = 0;
03264                             break;
03265                         }
03266                     if (found) {
03267                         if (iz > 0 && iz < 170)
03268                             met->pt[ix][iy] = (float) p2[iz];
03269                         break;
03270                     }
03271                 }
03272
03273                 /* Find 2nd tropopause... */
03274                 if (ctl->met_tropo == 4) {
03275                     met->pt[ix][iy] = GSL_NAN;
03276                     for (; iz <= 170; iz++) {
03277                         found = 1;
03278                         for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
03279                             if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
03280                                 * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) < 3.0) {
03281                                 found = 0;
03282                                 break;
03283                             }
03284                         if (found)
03285                             break;
03286                     }
03287                     for (; iz <= 170; iz++) {
03288                         found = 1;
03289                         for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
03290                             if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
03291                                 * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
03292                                 found = 0;
03293                                 break;
03294                             }
03295                         if (found) {
03296                             if (iz > 0 && iz < 170)

```

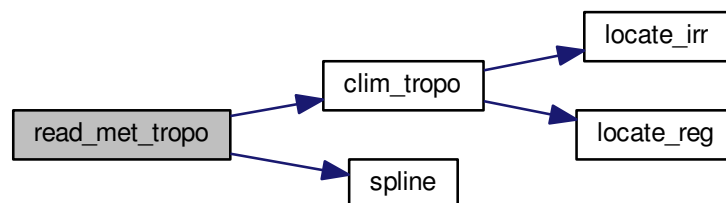


```

03297         met->pt[ix][iy] = (float) p2[iz];
03298         break;
03299     }
03300 }
03301 }
03302 }
03303 }
03304
03305 /* Use dynamical tropopause... */
03306 else if (ctl->met_tropo == 5) {
03307
03308     /* Loop over grid points... */
03309 #pragma omp parallel for default(shared) private(ix,iy,iz,pv,pv2,th,th2)
03310     for (ix = 0; ix < met->nx; ix++)
03311         for (iy = 0; iy < met->ny; iy++) {
03312
03313             /* Interpolate potential vorticity profile... */
03314             for (iz = 0; iz < met->np; iz++)
03315                 pv[iz] = met->pv[ix][iy][iz];
03316             spline(z, pv, met->np, z2, pv2, 171);
03317
03318             /* Interpolate potential temperature profile... */
03319             for (iz = 0; iz < met->np; iz++)
03320                 th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
03321             spline(z, th, met->np, z2, th2, 171);
03322
03323             /* Find dynamical tropopause 3.5 PVU + 380 K */
03324             met->pt[ix][iy] = GSL_NAN;
03325             for (iz = 0; iz <= 170; iz++)
03326                 if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
03327                     if (iz > 0 && iz < 170)
03328                         met->pt[ix][iy] = (float) p2[iz];
03329                     break;
03330                 }
03331         }
03332     }
03333
03334     else
03335         ERRMSG("Cannot calculate tropopause!");
03336 }

```

Here is the call graph for this function:



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

Read a control parameter from file or command line.

Definition at line 3340 of file libtrac.c.

```

03347     {
03348
03349     FILE *in = NULL;
03350
03351     char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],

```

```

03352     msg[2 * LEN], rvarname[LEN], rval[LEN];
03353
03354     int contain = 0, i;
03355
03356     /* Open file... */
03357     if (filename[strlen(filename) - 1] != '-')
03358         if (!(in = fopen(filename, "r")))
03359             ERRMSG("Cannot open file!");
03360
03361     /* Set full variable name... */
03362     if (arridx >= 0) {
03363         sprintf(fullname1, "%s[%d]", varname, arridx);
03364         sprintf(fullname2, "%s[*]", varname);
03365     } else {
03366         sprintf(fullname1, "%s", varname);
03367         sprintf(fullname2, "%s", varname);
03368     }
03369
03370     /* Read data... */
03371     if (in != NULL)
03372         while (fgets(line, LEN, in))
03373             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
03374                 if (strcasecmp(rvarname, fullname1) == 0 ||
03375                     strcasecmp(rvarname, fullname2) == 0) {
03376                     contain = 1;
03377                     break;
03378                 }
03379     for (i = 1; i < argc - 1; i++)
03380         if (strcasecmp(argv[i], fullname1) == 0 ||
03381             strcasecmp(argv[i], fullname2) == 0) {
03382             sprintf(rval, "%s", argv[i + 1]);
03383             contain = 1;
03384             break;
03385         }
03386
03387     /* Close file... */
03388     if (in != NULL)
03389         fclose(in);
03390
03391     /* Check for missing variables... */
03392     if (!contain) {
03393         if (strlen(defvalue) > 0)
03394             sprintf(rval, "%s", defvalue);
03395         else {
03396             sprintf(msg, "Missing variable %s!\n", fullname1);
03397             ERRMSG(msg);
03398         }
03399     }
03400
03401     /* Write info... */
03402     printf("%s = %s\n", fullname1, rval);
03403
03404     /* Return values... */
03405     if (value != NULL)
03406         sprintf(value, "%s", rval);
03407     return atof(rval);
03408 }

```

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

Spline interpolation.

Definition at line 3412 of file libtrac.c.

```

03418     {
03419
03420     gsl_interp_accel *acc;
03421
03422     gsl_spline *s;
03423
03424     /* Allocate... */
03425     acc = gsl_interp_accel_alloc();
03426     s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
03427
03428     /* Interpolate temperature profile... */
03429     gsl_spline_init(s, x, y, (size_t) n);
03430     for (int i = 0; i < n2; i++)
03431         if (x2[i] <= x[0])
03432             y2[i] = y[0];
03433         else if (x2[i] >= x[n - 1])

```

```
03434     y2[i] = y[n - 1];
03435     else
03436         y2[i] = gsl_spline_eval(s, x2[i], acc);
03437
03438     /* Free... */
03439     gsl_spline_free(s);
03440     gsl_interp_accel_free(acc);
03441 }
```

5.19.2.34 double stddev (double * data, int n)

Calculate standard deviation.

Definition at line 3445 of file libtrac.c.

```
03447     {
03448
03449     if (n <= 0)
03450         return 0;
03451
03452     double avg = 0, rms = 0;
03453
03454     for (int i = 0; i < n; ++i)
03455         avg += data[i];
03456     avg /= n;
03457
03458     for (int i = 0; i < n; ++i)
03459         rms += SQR(data[i] - avg);
03460
03461     return sqrt(rms / (n - 1));
03462 }
```

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

Convert date to seconds.

Definition at line 3466 of file libtrac.c.

```
03474     {
03475
03476     struct tm t0, t1;
03477
03478     t0.tm_year = 100;
03479     t0.tm_mon = 0;
03480     t0.tm_mday = 1;
03481     t0.tm_hour = 0;
03482     t0.tm_min = 0;
03483     t0.tm_sec = 0;
03484
03485     t1.tm_year = year - 1900;
03486     t1.tm_mon = mon - 1;
03487     t1.tm_mday = day;
03488     t1.tm_hour = hour;
03489     t1.tm_min = min;
03490     t1.tm_sec = sec;
03491
03492     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
03493 }
```

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

Measure wall-clock time.

Definition at line 3497 of file libtrac.c.

```

03500         {
03501
03502     static double starttime[NTIMER], runtime[NTIMER];
03503
03504     /* Check id... */
03505     if (id < 0 || id >= NTIMER)
03506         ERRMSG("Too many timers!");
03507
03508     /* Start timer... */
03509     if (mode == 1) {
03510         if (starttime[id] <= 0)
03511             starttime[id] = omp_get_wtime();
03512         else
03513             ERRMSG("Timer already started!");
03514     }
03515
03516     /* Stop timer... */
03517     else if (mode == 2) {
03518         if (starttime[id] > 0) {
03519             runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
03520             starttime[id] = -1;
03521         }
03522     }
03523
03524     /* Print timer... */
03525     else if (mode == 3) {
03526         printf("%s = %.3f s\n", name, runtime[id]);
03527         runtime[id] = 0;
03528     }
03529 }

```

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

Write atmospheric data.

Definition at line 3533 of file libtrac.c.

```

03537         {
03538
03539     FILE *in, *out;
03540
03541     char line[LEN];
03542
03543     double r, t0, t1;
03544
03545     int ip, iq, year, mon, day, hour, min, sec;
03546
03547     /* Set time interval for output... */
03548     t0 = t - 0.5 * ctl->dt_mod;
03549     t1 = t + 0.5 * ctl->dt_mod;
03550
03551     /* Write info... */
03552     printf("Write atmospheric data: %s\n", filename);
03553
03554     /* Write ASCII data... */
03555     if (ctl->atm_type == 0) {
03556
03557         /* Check if gnuplot output is requested... */
03558         if (ctl->atm_gpfile[0] != '-') {
03559
03560             /* Create gnuplot pipe... */
03561             if (!(out = popen("gnuplot", "w")))
03562                 ERRMSG("Cannot create pipe to gnuplot!");
03563
03564             /* Set plot filename... */
03565             fprintf(out, "set out \"%s.png\"\n", filename);
03566
03567             /* Set time string... */
03568             jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
03569             fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",

```

```

03570         year, mon, day, hour, min);
03571
03572     /* Dump gnuplot file to pipe... */
03573     if (!(in = fopen(ctl->atm_gpfile, "r")))
03574         ERRMSG("Cannot open file!");
03575     while (fgets(line, LEN, in))
03576         fprintf(out, "%s", line);
03577     fclose(in);
03578 }
03579
03580     else {
03581
03582         /* Create file... */
03583         if (!(out = fopen(filename, "w")))
03584             ERRMSG("Cannot create file!");
03585     }
03586
03587     /* Write header... */
03588     fprintf(out,
03589             "# $1 = time [s]\n"
03590             "# $2 = altitude [km]\n"
03591             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03592     for (iq = 0; iq < ctl->nq; iq++)
03593         fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
03594                 ctl->qnt_unit[iq]);
03595     fprintf(out, "\n");
03596
03597     /* Write data... */
03598     for (ip = 0; ip < atm->np; ip += ctl->atm_stride) {
03599
03600         /* Check time... */
03601         if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
03602             continue;
03603
03604         /* Write output... */
03605         fprintf(out, "%.2f %g %g", atm->time[ip], Z(atm->p[ip]),
03606                 atm->lon[ip], atm->lat[ip]);
03607         for (iq = 0; iq < ctl->nq; iq++) {
03608             fprintf(out, " ");
03609             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
03610         }
03611         fprintf(out, "\n");
03612     }
03613
03614     /* Close file... */
03615     fclose(out);
03616 }
03617
03618     /* Write binary data... */
03619     else if (ctl->atm_type == 1) {
03620
03621         /* Create file... */
03622         if (!(out = fopen(filename, "w")))
03623             ERRMSG("Cannot create file!");
03624
03625         /* Write data... */
03626         FWRITE(&atm->np, int,
03627              1,
03628              out);
03629         FWRITE(atm->time, double,
03630              (size_t) atm->np,
03631              out);
03632         FWRITE(atm->p, double,
03633              (size_t) atm->np,
03634              out);
03635         FWRITE(atm->lon, double,
03636              (size_t) atm->np,
03637              out);
03638         FWRITE(atm->lat, double,
03639              (size_t) atm->np,
03640              out);
03641         for (iq = 0; iq < ctl->nq; iq++)
03642             FWRITE(atm->q[iq], double,
03643                  (size_t) atm->np,
03644                  out);
03645
03646         /* Close file... */
03647         fclose(out);
03648     }
03649
03650     /* Error... */
03651     else
03652         ERRMSG("Atmospheric data type not supported!");
03653 }

```

Here is the call graph for this function:



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

Write CSI data.

Definition at line 3657 of file libtrac.c.

```

03661     {
03662
03663     static FILE *in, *out;
03664
03665     static char line[LEN];
03666
03667     static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
03668         rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
03669
03670     static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
03671
03672     /* Init... */
03673     if (t == ctl->t_start) {
03674
03675         /* Check quantity index for mass... */
03676         if (ctl->qnt_m < 0)
03677             ERRMSG("Need quantity mass!");
03678
03679         /* Open observation data file... */
03680         printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
03681         if (!(in = fopen(ctl->csi_obsfile, "r")))
03682             ERRMSG("Cannot open file!");
03683
03684         /* Create new file... */
03685         printf("Write CSI data: %s\n", filename);
03686         if (!(out = fopen(filename, "w")))
03687             ERRMSG("Cannot create file!");
03688
03689         /* Write header... */
03690         fprintf(out,
03691             "# $1 = time [s]\n"
03692             "# $2 = number of hits (cx)\n"
03693             "# $3 = number of misses (cy)\n"
03694             "# $4 = number of false alarms (cz)\n"
03695             "# $5 = number of observations (cx + cy)\n"
03696             "# $6 = number of forecasts (cx + cz)\n"
03697             "# $7 = bias (forecasts/observations) [%%]\n"
03698             "# $8 = probability of detection (POD) [%%]\n"
03699             "# $9 = false alarm rate (FAR) [%%]\n"
03700             "# $10 = critical success index (CSI) [%%]\n\n");
03701     }
03702
03703     /* Set time interval... */
03704     t0 = t - 0.5 * ctl->dt_mod;
03705     t1 = t + 0.5 * ctl->dt_mod;
03706
03707     /* Initialize grid cells... */
03708     #pragma omp parallel for default(shared) private(ix,iy,iz)
03709     for (ix = 0; ix < ctl->csi_nx; ix++)
03710         for (iy = 0; iy < ctl->csi_ny; iy++)
03711             for (iz = 0; iz < ctl->csi_nz; iz++)
03712                 modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
03713
03714     /* Read observation data... */
03715     while (fgets(line, LEN, in)) {
03716
03717         /* Read data... */

```

```

03718     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &rln, &rln) !=
03719         5)
03720         continue;
03721
03722     /* Check time... */
03723     if (rt < t0)
03724         continue;
03725     if (rt > t1)
03726         break;
03727
03728     /* Calculate indices... */
03729     ix = (int) ((rln - ctl->csi_lon0)
03730                / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
03731     iy = (int) ((rln - ctl->csi_lat0)
03732                / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
03733     iz = (int) ((rz - ctl->csi_z0)
03734                / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
03735
03736     /* Check indices... */
03737     if (ix < 0 || ix >= ctl->csi_nx ||
03738         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
03739         continue;
03740
03741     /* Get mean observation index... */
03742     obsmean[ix][iy][iz] += robs;
03743     obscount[ix][iy][iz]++;
03744 }
03745
03746 /* Analyze model data... */
03747 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
03748 for (ip = 0; ip < atm->np; ip++) {
03749
03750     /* Check time... */
03751     if (atm->time[ip] < t0 || atm->time[ip] > t1)
03752         continue;
03753
03754     /* Get indices... */
03755     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
03756                / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
03757     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
03758                / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
03759     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
03760                / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
03761
03762     /* Check indices... */
03763     if (ix < 0 || ix >= ctl->csi_nx ||
03764         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
03765         continue;
03766
03767     /* Get total mass in grid cell... */
03768     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
03769 }
03770
03771 /* Analyze all grid cells... */
03772 #pragma omp parallel for default(shared) private(ix,iy,iz,dlon,dlat,lat,area)
03773 for (ix = 0; ix < ctl->csi_nx; ix++)
03774     for (iy = 0; iy < ctl->csi_ny; iy++)
03775         for (iz = 0; iz < ctl->csi_nz; iz++) {
03776
03777             /* Calculate mean observation index... */
03778             if (obscount[ix][iy][iz] > 0)
03779                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
03780
03781             /* Calculate column density... */
03782             if (modmean[ix][iy][iz] > 0) {
03783                 dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
03784                 dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
03785                 lat = ctl->csi_lat0 + dlat * (iy + 0.5);
03786                 area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
03787                     * cos(lat * M_PI / 180.);
03788                 modmean[ix][iy][iz] /= (1e6 * area);
03789             }
03790
03791             /* Calculate CSI... */
03792             if (obscount[ix][iy][iz] > 0) {
03793                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
03794                     modmean[ix][iy][iz] >= ctl->csi_modmin)
03795                     cx++;
03796                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
03797                     modmean[ix][iy][iz] < ctl->csi_modmin)
03798                     cy++;
03799                 else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
03800                     modmean[ix][iy][iz] >= ctl->csi_modmin)
03801                     cz++;
03802             }
03803         }
03804

```

```

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

```

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

Write ensemble data.

Definition at line 3827 of file libtrac.c.

```

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

```

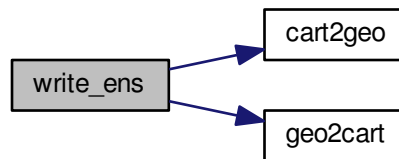


```

03885
03886     /* Write results... */
03887     if (n > 0) {
03888
03889         /* Get mean position... */
03890         xm[0] = xm[1] = xm[2] = 0;
03891         for (i = 0; i < n; i++) {
03892             xm[0] += x[i][0] / (double) n;
03893             xm[1] += x[i][1] / (double) n;
03894             xm[2] += x[i][2] / (double) n;
03895         }
03896         cart2geo(xm, &dummy, &lon, &lat);
03897         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
03898             lat);
03899
03900         /* Get quantity statistics... */
03901         for (iq = 0; iq < ctl->nq; iq++) {
03902             fprintf(out, " ");
03903             fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
03904         }
03905         for (iq = 0; iq < ctl->nq; iq++) {
03906             fprintf(out, " ");
03907             fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
03908         }
03909         fprintf(out, " %lu\n", n);
03910     }
03911
03912     /* Init new ensemble... */
03913     ens = atm->q[ctl->qnt_ens][ip];
03914     n = 0;
03915 }
03916
03917 /* Save data... */
03918 p[n] = atm->p[ip];
03919 geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
03920 for (iq = 0; iq < ctl->nq; iq++)
03921     q[iq][n] = atm->q[iq][ip];
03922 if ((++n) >= NENS)
03923     ERRMSG("Too many data points!");
03924 }
03925
03926 /* Write results... */
03927 if (n > 0) {
03928
03929     /* Get mean position... */
03930     xm[0] = xm[1] = xm[2] = 0;
03931     for (i = 0; i < n; i++) {
03932         xm[0] += x[i][0] / (double) n;
03933         xm[1] += x[i][1] / (double) n;
03934         xm[2] += x[i][2] / (double) n;
03935     }
03936     cart2geo(xm, &dummy, &lon, &lat);
03937     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
03938
03939     /* Get quantity statistics... */
03940     for (iq = 0; iq < ctl->nq; iq++) {
03941         fprintf(out, " ");
03942         fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
03943     }
03944     for (iq = 0; iq < ctl->nq; iq++) {
03945         fprintf(out, " ");
03946         fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
03947     }
03948     fprintf(out, " %lu\n", n);
03949 }
03950
03951 /* Close file... */
03952 if (t == ctl->t_stop)
03953     fclose(out);
03954 }

```

Here is the call graph for this function:



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

Write gridded data.

Definition at line 3958 of file libtrac.c.

```

03964     {
03965
03966     FILE *in, *out;
03967
03968     char line[LEN];
03969
03970     static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
03971     area, rho_air, press, temp, cd, vmr, t0, t1, r, cw[3];
03972
03973     static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec,
03974     ci[3];
03975
03976     /* Check dimensions... */
03977     if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
03978         ERRMSG("Grid dimensions too large!");
03979
03980     /* Set time interval for output... */
03981     t0 = t - 0.5 * ctl->dt_mod;
03982     t1 = t + 0.5 * ctl->dt_mod;
03983
03984     /* Set grid box size... */
03985     dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
03986     dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
03987     dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
03988
03989     /* Initialize grid... */
03990 #pragma omp parallel for default(shared) private(ix,iy,iz)
03991     for (ix = 0; ix < ctl->grid_nx; ix++)
03992         for (iy = 0; iy < ctl->grid_ny; iy++)
03993             for (iz = 0; iz < ctl->grid_nz; iz++) {
03994                 mass[ix][iy][iz] = 0;
03995                 np[ix][iy][iz] = 0;
03996             }
03997
03998     /* Average data... */
03999 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
04000     for (ip = 0; ip < atm->np; ip++)
04001         if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
04002
04003             /* Get index... */
04004             ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
04005             iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
04006             iz = (int) ((atm->p[ip] - ctl->grid_z0) / dz);
04007
04008             /* Check indices... */
04009             if (ix < 0 || ix >= ctl->grid_nx ||
04010                 iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
04011                 continue;
04012
04013             /* Add mass... */
04014             if (ctl->qnt_m >= 0)
04015                 mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
  
```

```

04016     np[ix][iy][iz]++;
04017 }
04018
04019 /* Check if gnuplot output is requested... */
04020 if (ctl->grid_gpfile[0] != '-') {
04021
04022     /* Write info... */
04023     printf("Plot grid data: %s.png\n", filename);
04024
04025     /* Create gnuplot pipe... */
04026     if (!(out = popen("gnuplot", "w")))
04027         ERRMSG("Cannot create pipe to gnuplot!");
04028
04029     /* Set plot filename... */
04030     fprintf(out, "set out \"%s.png\"\n", filename);
04031
04032     /* Set time string... */
04033     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
04034     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
04035             year, mon, day, hour, min);
04036
04037     /* Dump gnuplot file to pipe... */
04038     if (!(in = fopen(ctl->grid_gpfile, "r")))
04039         ERRMSG("Cannot open file!");
04040     while (fgets(line, LEN, in))
04041         fprintf(out, "%s", line);
04042     fclose(in);
04043 }
04044
04045 else {
04046
04047     /* Write info... */
04048     printf("Write grid data: %s\n", filename);
04049
04050     /* Create file... */
04051     if (!(out = fopen(filename, "w")))
04052         ERRMSG("Cannot create file!");
04053 }
04054
04055 /* Write header... */
04056 fprintf(out,
04057         "# $1 = time [s]\n"
04058         "# $2 = altitude [km]\n"
04059         "# $3 = longitude [deg]\n"
04060         "# $4 = latitude [deg]\n"
04061         "# $5 = surface area [km^2]\n"
04062         "# $6 = layer width [km]\n"
04063         "# $7 = number of particles [l]\n"
04064         "# $8 = column density [kg/m^2]\n"
04065         "# $9 = volume mixing ratio [ppv]\n\n");
04066
04067 /* Write data... */
04068 for (ix = 0; ix < ctl->grid_nx; ix++) {
04069     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
04070         fprintf(out, "\n");
04071     for (iy = 0; iy < ctl->grid_ny; iy++) {
04072         if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
04073             fprintf(out, "\n");
04074         for (iz = 0; iz < ctl->grid_nz; iz++)
04075             if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
04076
04077                 /* Set coordinates... */
04078                 z = ctl->grid_z0 + dz * (iz + 0.5);
04079                 lon = ctl->grid_lon0 + dlon * (ix + 0.5);
04080                 lat = ctl->grid_lat0 + dlat * (iy + 0.5);
04081
04082                 /* Get pressure and temperature... */
04083                 press = P(z);
04084                 intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press, lon,
04085                                 lat, &temp, ci, cw, 1);
04086
04087                 /* Calculate surface area... */
04088                 area = dlat * dlon * SQR(RE * M_PI / 180.)
04089                     * cos(lat * M_PI / 180.);
04090
04091                 /* Calculate column density... */
04092                 cd = mass[ix][iy][iz] / (1e6 * area);
04093
04094                 /* Calculate volume mixing ratio... */
04095                 rho_air = 100. * press / (RA * temp);
04096                 vmr = (ctl->molmass > 0) ? MA / ctl->molmass * mass[ix][iy][iz]
04097                     / (rho_air * 1e6 * area * 1e3 * dz) : GSL_NAN;
04098
04099                 /* Write output... */
04100                 fprintf(out, "%.2f %g %g %g %g %d %g %g\n",
04101                         t, z, lon, lat, area, dz, np[ix][iy][iz], cd, vmr);
04102             }

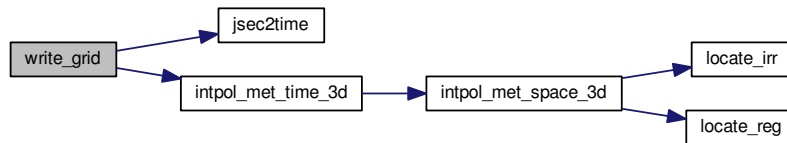
```

```

04103     }
04104 }
04105
04106 /* Close file... */
04107 fclose(out);
04108 }

```

Here is the call graph for this function:



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

Write profile data.

Definition at line 4112 of file libtrac.c.

```

04118     {
04119
04120     static FILE *in, *out;
04121
04122     static char line[LEN];
04123
04124     static double mass[GX][GY][GZ], obsmean[GX][GY], rt, rz, rlon, rlat, robs,
04125         t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, vmr, h2o,
04126         o3, cw[3];
04127
04128     static int obscount[GX][GY], ip, ix, iy, iz, okay, ci[3];
04129
04130     /* Init... */
04131     if (t == ctl->t_start) {
04132
04133         /* Check quantity index for mass... */
04134         if (ctl->qnt_m < 0)
04135             ERRMSG("Need quantity mass!");
04136
04137         /* Check dimensions... */
04138         if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
04139             ERRMSG("Grid dimensions too large!");
04140
04141         /* Check molar mass... */
04142         if (ctl->molmass <= 0)
04143             ERRMSG("Specify molar mass!");
04144
04145         /* Open observation data file... */
04146         printf("Read profile observation data: %s\n", ctl->prof_obsfile);
04147         if (!(in = fopen(ctl->prof_obsfile, "r")))
04148             ERRMSG("Cannot open file!");
04149
04150         /* Create new output file... */
04151         printf("Write profile data: %s\n", filename);
04152         if (!(out = fopen(filename, "w")))
04153             ERRMSG("Cannot create file!");
04154
04155         /* Write header... */
04156         fprintf(out,
04157             "# $1 = time [s]\n"
04158             "# $2 = altitude [km]\n"
04159             "# $3 = longitude [deg]\n"
04160             "# $4 = latitude [deg]\n"
04161             "# $5 = pressure [hPa]\n"
04162             "# $6 = temperature [K]\n"
04163             "# $7 = volume mixing ratio [ppv]\n"
04164             "# $8 = H2O volume mixing ratio [ppv]\n"

```

```

04165         "# $9 = O3 volume mixing ratio [ppv]\n"
04166         "# $10 = observed BT index [K]\n");
04167
04168     /* Set grid box size... */
04169     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
04170     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
04171     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
04172 }
04173
04174 /* Set time interval... */
04175 t0 = t - 0.5 * ctl->dt_mod;
04176 t1 = t + 0.5 * ctl->dt_mod;
04177
04178 /* Initialize... */
04179 #pragma omp parallel for default(shared) private(ix,iy,iz)
04180 for (ix = 0; ix < ctl->prof_nx; ix++)
04181     for (iy = 0; iy < ctl->prof_ny; iy++) {
04182         obsmean[ix][iy] = 0;
04183         obscount[ix][iy] = 0;
04184         for (iz = 0; iz < ctl->prof_nz; iz++)
04185             mass[ix][iy][iz] = 0;
04186     }
04187
04188 /* Read observation data... */
04189 while (fgets(line, LEN, in)) {
04190
04191     /* Read data... */
04192     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &robs) !=
04193         5)
04194         continue;
04195
04196     /* Check time... */
04197     if (rt < t0)
04198         continue;
04199     if (rt > t1)
04200         break;
04201
04202     /* Calculate indices... */
04203     ix = (int) ((rln - ctl->prof_lon0) / dlon);
04204     iy = (int) ((rln - ctl->prof_lat0) / dlat);
04205
04206     /* Check indices... */
04207     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
04208         continue;
04209
04210     /* Get mean observation index... */
04211     obsmean[ix][iy] += robs;
04212     obscount[ix][iy]++;
04213 }
04214
04215 /* Analyze model data... */
04216 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
04217 for (ip = 0; ip < atm->np; ip++) {
04218
04219     /* Check time... */
04220     if (atm->time[ip] < t0 || atm->time[ip] > t1)
04221         continue;
04222
04223     /* Get indices... */
04224     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
04225     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
04226     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
04227
04228     /* Check indices... */
04229     if (ix < 0 || ix >= ctl->prof_nx ||
04230         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
04231         continue;
04232
04233     /* Get total mass in grid cell... */
04234     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04235 }
04236
04237 /* Extract profiles... */
04238 for (ix = 0; ix < ctl->prof_nx; ix++)
04239     for (iy = 0; iy < ctl->prof_ny; iy++)
04240         if (obscount[ix][iy] > 0) {
04241
04242             /* Check profile... */
04243             okay = 0;
04244             for (iz = 0; iz < ctl->prof_nz; iz++)
04245                 if (mass[ix][iy][iz] > 0) {
04246                     okay = 1;
04247                     break;
04248                 }
04249             if (!okay)
04250                 continue;
04251

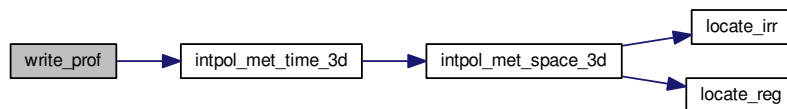
```

```

04252      /* Write output... */
04253      fprintf(out, "\n");
04254
04255      /* Loop over altitudes... */
04256      for (iz = 0; iz < ctl->prof_nz; iz++) {
04257
04258          /* Set coordinates... */
04259          z = ctl->prof_z0 + dz * (iz + 0.5);
04260          lon = ctl->prof_lon0 + dlon * (ix + 0.5);
04261          lat = ctl->prof_lat0 + dlat * (iy + 0.5);
04262
04263          /* Get pressure and temperature... */
04264          press = P(z);
04265          intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press, lon,
04266                          lat, &temp, ci, cw, 1);
04267          intpol_met_time_3d(met0, met0->h2o, met1, met1->
04268                          h2o, t, press, lon,
04269                          lat, &h2o, ci, cw, 0);
04269          intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press, lon,
04270                          lat, &o3, ci, cw, 0);
04271
04272          /* Calculate surface area... */
04273          area = dlat * dlon * SQR(M_PI * RE / 180.)
04274              * cos(lat * M_PI / 180.);
04275
04276          /* Calculate volume mixing ratio... */
04277          rho_air = 100. * press / (RA * temp);
04278          vmr = MA / ctl->molmass * mass[ix][iy][iz]
04279              / (rho_air * area * dz * 1e9);
04280
04281          /* Write output... */
04282          fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
04283                  t, z, lon, lat, press, temp, vmr, h2o, o3,
04284                  obsmean[ix][iy] / obscount[ix][iy]);
04285      }
04286  }
04287
04288      /* Close file... */
04289      if (t == ctl->t_stop)
04290          fclose(out);
04291  }

```

Here is the call graph for this function:



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

Write station data.

Definition at line 4295 of file libtrac.c.

```

04299      {
04300
04301          static FILE *out;
04302
04303          static double rmax2, t0, t1, x0[3], x1[3];
04304
04305          /* Init... */
04306          if (t == ctl->t_start) {
04307
04308              /* Write info... */
04309              printf("Write station data: %s\n", filename);
04310
04311              /* Create new file... */

```

```

04312     if (!(out = fopen(filename, "w")))
04313         ERRMSG("Cannot create file!");
04314
04315     /* Write header... */
04316     fprintf(out,
04317             "# $1 = time [s]\n"
04318             "# $2 = altitude [km]\n"
04319             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
04320     for (int iq = 0; iq < ctl->nq; iq++)
04321         fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
04322                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
04323     fprintf(out, "\n");
04324
04325     /* Set geolocation and search radius... */
04326     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
04327     rmax2 = SQR(ctl->stat_r);
04328 }
04329
04330 /* Set time interval for output... */
04331 t0 = t - 0.5 * ctl->dt_mod;
04332 t1 = t + 0.5 * ctl->dt_mod;
04333
04334 /* Loop over air parcels... */
04335 for (int ip = 0; ip < atm->np; ip++) {
04336
04337     /* Check time... */
04338     if (atm->time[ip] < t0 || atm->time[ip] > t1)
04339         continue;
04340
04341     /* Check station flag... */
04342     if (ctl->qnt_stat >= 0)
04343         if (atm->q[ctl->qnt_stat][ip])
04344             continue;
04345
04346     /* Get Cartesian coordinates... */
04347     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
04348
04349     /* Check horizontal distance... */
04350     if (DIST2(x0, x1) > rmax2)
04351         continue;
04352
04353     /* Set station flag... */
04354     if (ctl->qnt_stat >= 0)
04355         atm->q[ctl->qnt_stat][ip] = 1;
04356
04357     /* Write data... */
04358     fprintf(out, "%.2f %g %g %g",
04359             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
04360     for (int iq = 0; iq < ctl->nq; iq++) {
04361         fprintf(out, " ");
04362         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
04363     }
04364     fprintf(out, "\n");
04365 }
04366
04367 /* Close file... */
04368 if (t == ctl->t_stop)
04369     fclose(out);
04370 }

```

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  static double clim_hno3_secs[12] = {
00044      1209600.00, 3888000.00, 6393600.00,
00045      9072000.00, 11664000.00, 14342400.00,
00046      16934400.00, 19612800.00, 22291200.00,
00047      24883200.00, 27561600.00, 30153600.00
00048  };
00049
00050  #ifdef _OPENACC
00051  #pragma acc declare copyin(clim_hno3_secs)
00052  #endif
00053
00054  static double clim_hno3_lats[18] = {
00055      -85, -75, -65, -55, -45, -35, -25, -15, -5,
00056      5, 15, 25, 35, 45, 55, 65, 75, 85
00057  };
00058
00059  #ifdef _OPENACC
00060  #pragma acc declare copyin(clim_hno3_lats)
00061  #endif
00062
00063  static double clim_hno3_ps[10] = {
00064      4.64159, 6.81292, 10, 14.678, 21.5443,
00065      31.6228, 46.4159, 68.1292, 100, 146.78
00066  };
00067
00068  #ifdef _OPENACC
00069  #pragma acc declare copyin(clim_hno3_ps)
00070  #endif
00071
00072  static double clim_hno3_var[12][18][10] = {
00073      {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00074       {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00075       {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
00076       {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00077       {0.818, 1.62, 2.77, 4.52, 6.28, 6.84, 5.83, 3.05, 1.15, 0.709},
00078       {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
00079       {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00080       {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
00081       {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
00082       {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00083       {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
00084       {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00085       {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
00086       {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00087       {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00088       {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00089       {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
00090       {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
00091      {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00092       {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
00093       {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},

```



```

00094 {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00095 {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
00096 {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
00097 {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
00098 {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
00099 {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
00100 {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
00101 {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00102 {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
00103 {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
00104 {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
00105 {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
00106 {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
00107 {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
00108 {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17},
00109 {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00110 {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
00111 {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
00112 {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
00113 {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
00114 {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
00115 {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
00116 {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
00117 {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
00118 {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
00119 {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
00120 {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
00121 {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
00122 {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
00123 {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
00124 {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
00125 {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
00126 {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42},
00127 {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
00128 {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00129 {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
00130 {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
00131 {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
00132 {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
00133 {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
00134 {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
00135 {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
00136 {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
00137 {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
00138 {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
00139 {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
00140 {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
00141 {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
00142 {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
00143 {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
00144 {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62},
00145 {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
00146 {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
00147 {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
00148 {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
00149 {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
00150 {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
00151 {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
00152 {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
00153 {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
00154 {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
00155 {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
00156 {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
00157 {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
00158 {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
00159 {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
00160 {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
00161 {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
00162 {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6},
00163 {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
00164 {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
00165 {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
00166 {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
00167 {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
00168 {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
00169 {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
00170 {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
00171 {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
00172 {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
00173 {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
00174 {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
00175 {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
00176 {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
00177 {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
00178 {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
00179 {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
00180 {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91},

```

```

00181  {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33}},
00182  {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78}},
00183  {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08}},
00184  {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3}},
00185  {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38}},
00186  {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656}},
00187  {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176}},
00188  {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705}},
00189  {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12}},
00190  {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199}},
00191  {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25}},
00192  {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259}},
00193  {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422}},
00194  {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913}},
00195  {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4}},
00196  {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56}},
00197  {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61}},
00198  {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}},
00199  {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4}},
00200  {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73}},
00201  {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6}},
00202  {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14}},
00203  {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38}},
00204  {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672}},
00205  {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19}},
00206  {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181}},
00207  {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107}},
00208  {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185}},
00209  {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224}},
00210  {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232}},
00211  {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341}},
00212  {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754}},
00213  {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23}},
00214  {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45}},
00215  {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5}},
00216  {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
00217  {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56}},
00218  {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74}},
00219  {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09}},
00220  {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83}},
00221  {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22}},
00222  {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646}},
00223  {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169}},
00224  {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587}},
00225  {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815}},
00226  {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147}},
00227  {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197}},
00228  {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163}},
00229  {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303}},
00230  {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714}},
00231  {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1}},
00232  {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41}},
00233  {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56}},
00234  {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
00235  {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91}},
00236  {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84}},
00237  {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97}},
00238  {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56}},
00239  {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11}},
00240  {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616}},
00241  {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21}},
00242  {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968}},
00243  {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105}},
00244  {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146}},
00245  {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178}},
00246  {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14}},
00247  {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353}},
00248  {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802}},
00249  {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2}},
00250  {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52}},
00251  {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73}},
00252  {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
00253  {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78}},
00254  {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73}},
00255  {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75}},
00256  {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41}},
00257  {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955}},
00258  {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61}},
00259  {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269}},
00260  {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132}},
00261  {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121}},
00262  {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147}},
00263  {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146}},
00264  {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172}},
00265  {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448}},
00266  {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948}},
00267  {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56}},

```

```

00268     {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
00269     {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
00270     {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
00271     {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
00272     {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
00273     {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
00274     {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
00275     {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
00276     {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
00277     {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
00278     {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
00279     {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
00280     {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
00281     {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
00282     {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
00283     {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
00284     {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
00285     {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
00286     {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
00287     {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
00288     {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
00289 };
00290
00291 #ifdef _OPENACC
00292 #pragma acc declare copyin(clim_hno3_var)
00293 #endif
00294
00295 double clim_hno3(
00296     double t,
00297     double lat,
00298     double p) {
00299
00300     /* Get seconds since begin of year... */
00301     double sec = FMOD(t, 365.25 * 86400.);
00302     while (sec < 0)
00303         sec += 365.25 * 86400.;
00304
00305     /* Check pressure... */
00306     if (p < clim_hno3_ps[0])
00307         p = clim_hno3_ps[0];
00308     else if (p > clim_hno3_ps[9])
00309         p = clim_hno3_ps[9];
00310
00311     /* Get indices... */
00312     int isec = locate_irr(clim_hno3_secs, 12, sec);
00313     int ilat = locate_reg(clim_hno3_lats, 18, lat);
00314     int ip = locate_irr(clim_hno3_ps, 10, p);
00315
00316     /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
00317     double aux00 = LIN(clim_hno3_ps[ip],
00318                       clim_hno3_var[isec][ilat][ip],
00319                       clim_hno3_ps[ip + 1],
00320                       clim_hno3_var[isec][ilat][ip + 1], p);
00321     double aux01 = LIN(clim_hno3_ps[ip],
00322                       clim_hno3_var[isec][ilat + 1][ip],
00323                       clim_hno3_ps[ip + 1],
00324                       clim_hno3_var[isec][ilat + 1][ip + 1], p);
00325     double aux10 = LIN(clim_hno3_ps[ip],
00326                       clim_hno3_var[isec + 1][ilat][ip],
00327                       clim_hno3_ps[ip + 1],
00328                       clim_hno3_var[isec + 1][ilat][ip + 1], p);
00329     double aux11 = LIN(clim_hno3_ps[ip],
00330                       clim_hno3_var[isec + 1][ilat + 1][ip],
00331                       clim_hno3_ps[ip + 1],
00332                       clim_hno3_var[isec + 1][ilat + 1][ip + 1], p);
00333     aux00 = LIN(clim_hno3_lats[ilat], aux00,
00334               clim_hno3_lats[ilat + 1], aux01, lat);
00335     aux11 = LIN(clim_hno3_lats[ilat], aux10,
00336               clim_hno3_lats[ilat + 1], aux11, lat);
00337     return LIN(clim_hno3_secs[isec], aux00,
00338               clim_hno3_secs[isec + 1], aux11, sec);
00339 }
00340
00341 /*****
00342
00343 static double clim_oh_secs[12] = {
00344     1209600.00, 3888000.00, 6393600.00,
00345     9072000.00, 11664000.00, 14342400.00,
00346     16934400.00, 19612800.00, 22291200.00,
00347     24883200.00, 27561600.00, 30153600.00
00348 };
00349
00350 #ifdef _OPENACC
00351 #pragma acc declare copyin(clim_oh_secs)
00352 #endif
00353
00354 static double clim_oh_lats[18] = {

```

```

00355     -85, -75, -65, -55, -45, -35, -25, -15, -5,
00356     5, 15, 25, 35, 45, 55, 65, 75, 85
00357 };
00358
00359 #ifndef _OPENACC
00360 #pragma acc declare copyin(clim_oh_lats)
00361 #endif
00362
00363 static double clim_oh_ps[34] = {
00364     0.17501, 0.233347, 0.31113, 0.41484, 0.553119, 0.737493, 0.983323,
00365     1.3111, 1.74813, 2.33084, 3.10779, 4.14372, 5.52496, 7.36661, 9.82214,
00366     13.0962, 17.4616, 23.2821, 31.0428, 41.3904, 55.1872, 73.583, 98.1107,
00367     130.814, 174.419, 232.559, 310.078, 413.438, 551.25, 735, 789.809,
00368     848.705, 911.993, 980
00369 };
00370
00371 #ifndef _OPENACC
00372 #pragma acc declare copyin(clim_oh_ps)
00373 #endif
00374
00375 static double clim_oh_var[12][18][34] = {
00376     {{6.422, 6.418, 7.221, 8.409, 9.768, 11.22, 12.65, 13.68, 14.03,
00377       13.06, 11.01, 8.791, 7.096, 6.025, 5.135, 4.057, 2.791, 1.902,
00378       1.318, 0.9553, 0.7083, 0.5542, 0.5145, 0.5485, 0.6292, 0.5982, 1.716,
00379       1.111, 0.9802, 0.6707, 0.5235, 0.4476, 0.3783, 0.3091}},
00380     {{6.311, 6.394, 7.2, 8.349, 9.664, 11.02, 12.21, 13.06, 13.28,
00381       12.42, 10.59, 8.552, 6.944, 5.862, 4.948, 3.826, 2.689, 1.873,
00382       1.302, 0.9316, 0.7053, 0.5634, 0.508, 0.5207, 0.6166, 0.6789, 1.682,
00383       1.218, 1.079, 0.7621, 0.6662, 0.5778, 0.4875, 0.3997}},
00384     {{5.851, 5.827, 6.393, 7.294, 8.322, 9.415, 10.46, 11.24, 11.59,
00385       11.13, 9.754, 7.97, 6.417, 5.331, 4.468, 3.512, 2.581, 1.855,
00386       1.336, 0.9811, 0.756, 0.6328, 0.6011, 0.6202, 0.7603, 0.8883, 1.303,
00387       1.124, 1.118, 0.9428, 0.8655, 0.8156, 0.7602, 0.6805}},
00388     {{5.276, 5.158, 5.66, 6.463, 7.419, 8.488, 9.563, 10.45, 10.94,
00389       10.65, 9.465, 7.762, 6.204, 5.074, 4.209, 3.324, 2.511, 1.865,
00390       1.386, 1.066, 0.8521, 0.723, 0.6997, 0.7492, 0.8705, 0.8088, 1.22,
00391       1.192, 1.298, 1.096, 1.037, 0.9589, 0.8856, 0.7726}},
00392     {{5.06, 4.919, 5.379, 6.142, 7.095, 8.156, 9.18, 10.09, 10.62,
00393       10.33, 9.123, 7.479, 5.967, 4.858, 3.987, 3.097, 2.342, 1.743,
00394       1.323, 1.044, 0.8598, 0.7596, 0.7701, 0.7858, 0.8741, 1.256, 1.266,
00395       1.418, 1.594, 1.247, 1.169, 1.111, 1.054, 0.9141}},
00396     {{4.921, 4.759, 5.188, 5.936, 6.847, 7.871, 8.903, 9.805, 10.31,
00397       10, 8.818, 7.223, 5.757, 4.66, 3.75, 2.831, 2.1, 1.579,
00398       1.243, 1.017, 0.8801, 0.8193, 0.9409, 1.131, 0.7313, 1.201, 1.383,
00399       1.643, 1.751, 1.494, 1.499, 1.647, 1.934, 2.147}},
00400     {{4.665, 4.507, 4.947, 5.652, 6.549, 7.573, 8.609, 9.499, 9.985,
00401       9.664, 8.478, 6.944, 5.519, 4.407, 3.511, 2.595, 1.917, 1.46,
00402       1.172, 1.009, 0.9372, 0.9439, 1.047, 1.219, 0.5712, 1.032, 1.342,
00403       1.716, 1.846, 1.551, 1.55, 1.686, 2.006, 2.235}},
00404     {{4.424, 4.288, 4.678, 5.38, 6.271, 7.291, 8.324, 9.231, 9.678,
00405       9.264, 8.037, 6.532, 5.141, 4.037, 3.148, 2.319, 1.715, 1.318,
00406       1.078, 0.9647, 0.9327, 0.9604, 1.023, 0.4157, 0.4762, 1.04, 1.589,
00407       2.093, 1.957, 1.557, 1.52, 1.565, 1.776, 1.904}},
00408     {{4.154, 3.996, 4.347, 5.004, 5.854, 6.869, 7.929, 8.837, 9.23,
00409       8.708, 7.447, 6.024, 4.761, 3.742, 2.898, 2.096, 1.55, 1.191,
00410       0.9749, 0.8889, 0.8745, 0.9004, 0.9648, 0.36, 0.4423, 0.973, 1.571,
00411       2.086, 1.971, 1.569, 1.537, 1.567, 1.74, 1.811}},
00412     {{3.862, 3.738, 4.093, 4.693, 5.499, 6.481, 7.489, 8.328, 8.637,
00413       8.07, 6.863, 5.56, 4.438, 3.522, 2.736, 1.971, 1.441, 1.098,
00414       0.8945, 0.8155, 0.7965, 0.8013, 0.8582, 1.119, 0.4076, 0.8805, 1.446,
00415       1.977, 1.96, 1.713, 1.793, 2.055, 2.521, 2.776}},
00416     {{3.619, 3.567, 3.943, 4.54, 5.295, 6.168, 7.033, 7.691, 7.884,
00417       7.326, 6.207, 5.032, 4.055, 3.263, 2.552, 1.871, 1.365, 1.022,
00418       0.8208, 0.7184, 0.6701, 0.6551, 0.6965, 0.7928, 0.3639, 0.6365, 0.9295,
00419       1.381, 1.847, 1.658, 1.668, 1.87, 2.245, 2.409}},
00420     {{3.354, 3.395, 3.811, 4.39, 5.07, 5.809, 6.514, 7, 7.054,
00421       6.472, 5.463, 4.466, 3.649, 2.997, 2.396, 1.785, 1.289, 0.9304,
00422       0.7095, 0.5806, 0.5049, 0.4639, 0.4899, 0.5149, 0.5445, 0.5185, 0.7495,
00423       0.8662, 1.25, 1.372, 1.384, 1.479, 1.76, 1.874}},
00424     {{3.008, 3.102, 3.503, 4.049, 4.657, 5.287, 5.845, 6.14, 6.032,
00425       5.401, 4.494, 3.665, 3.043, 2.575, 2.103, 1.545, 1.074, 0.7429,
00426       0.5514, 0.4313, 0.3505, 0.2957, 0.2688, 0.2455, 0.232, 0.3565, 0.4017,
00427       0.5063, 0.6618, 0.7621, 0.7915, 0.8372, 0.923, 0.9218}},
00428     {{2.548, 2.725, 3.135, 3.637, 4.165, 4.666, 5.013, 5.056, 4.72,
00429       4.033, 3.255, 2.64, 2.24, 1.942, 1.555, 1.085, 0.7271, 0.502,
00430       0.3748, 0.2897, 0.2303, 0.19, 0.1645, 0.1431, 0.1215, 0.09467, 0.1442,
00431       0.1847, 0.2368, 0.2463, 0.2387, 0.2459, 0.2706, 0.2751}},
00432     {{1.946, 2.135, 2.46, 2.831, 3.203, 3.504, 3.584, 3.37, 2.921,
00433       2.357, 1.865, 1.551, 1.392, 1.165, 0.8443, 0.5497, 0.3686, 0.2632,
00434       0.1978, 0.1509, 0.1197, 0.0992, 0.08402, 0.07068, 0.05652, 0.03962,
00435       0.03904,
00436       0.04357, 0.05302, 0.04795, 0.04441, 0.04296, 0.04446, 0.04576}},
00437     {{1.157, 1.285, 1.432, 1.546, 1.602, 1.556, 1.378, 1.15, 0.9351,
00438       0.7636, 0.6384, 0.5267, 0.4008, 0.2821, 0.2, 0.1336, 0.09109, 0.06557,
00439       0.05219, 0.04197, 0.03443, 0.03119, 0.02893, 0.02577, 0.02119, 0.01102,
00440       0.008897,
00441       0.00467, 0.004651, 0.004809, 0.004539, 0.004181, 0.003737, 0.002833}},

```

```

00442 {0.07716, 0.06347, 0.05343, 0.04653, 0.0393, 0.03205, 0.02438, 0.01692,
00443 0.0115,
00444 0.007576, 0.00488, 0.002961, 0.001599, 0.001033, 0.001067, 0.001091,
00445 0.0005156, 0.0003818,
00446 0.0005061, 0.0005322, 0.0008027, 0.0008598, 0.0009114, 0.001112, 0.002042,
00447 0.0002528, 0.0005562,
00448 7.905e-06, 1.461e-07, 1.448e-07, 9.962e-08, 4.304e-08, 9.129e-17,
00449 1.36e-16},
00450 {2.613e-05, 3.434e-05, 3.646e-05, 5.101e-05, 8.027e-05, 0.0001172,
00451 9.886e-05, 1.933e-05, 3.14e-05,
00452 7.708e-05, 0.000136, 0.0001447, 0.0001049, 4.451e-05, 9.37e-06, 2.235e-06,
00453 1.034e-06, 4.87e-06,
00454 1.615e-05, 2.018e-05, 6.578e-05, 0.000178, 0.0002489, 0.0004818, 0.001231,
00455 0.0001402, 0.0004263,
00456 4.581e-07, 1.045e-12, 1.295e-13, 9.008e-14, 8.464e-14, 1.183e-13,
00457 2.471e-13}},
00458 {{5.459, 5.793, 6.743, 7.964, 9.289, 10.5, 11.39, 11.68, 11.14,
00459 9.663, 7.886, 6.505, 5.549, 4.931, 4.174, 3.014, 1.99, 1.339,
00460 0.9012, 0.6096, 0.4231, 0.3152, 0.2701, 0.2561, 0.2696, 0.2523, 0.7171,
00461 0.5333, 0.4876, 0.3218, 0.2536, 0.2178, 0.1861, 0.1546},
00462 {5.229, 5.456, 6.192, 7.112, 8.094, 9.038, 9.776, 10.16, 9.992,
00463 9, 7.493, 6.162, 5.154, 4.461, 3.788, 2.935, 2.058, 1.407,
00464 0.9609, 0.6738, 0.4989, 0.3927, 0.3494, 0.3375, 0.3689, 0.3866, 0.6716,
00465 0.7088, 0.6307, 0.4388, 0.3831, 0.3318, 0.2801, 0.2317},
00466 {4.712, 4.75, 5.283, 6.062, 6.943, 7.874, 8.715, 9.344, 9.516,
00467 8.913, 7.63, 6.223, 5.1, 4.346, 3.709, 2.982, 2.235, 1.605,
00468 1.142, 0.8411, 0.6565, 0.5427, 0.4942, 0.4907, 0.5447, 0.6331, 0.9356,
00469 0.7821, 0.7611, 0.663, 0.628, 0.5915, 0.5763, 0.5451},
00470 {4.621, 4.6, 5.091, 5.827, 6.688, 7.624, 8.529, 9.276, 9.631,
00471 9.219, 7.986, 6.499, 5.264, 4.401, 3.737, 2.996, 2.292, 1.69,
00472 1.237, 0.9325, 0.7325, 0.6093, 0.5742, 0.5871, 0.6446, 0.6139, 0.9845,
00473 0.9741, 1.044, 0.9091, 0.8661, 0.8218, 0.7617, 0.6884},
00474 {4.647, 4.573, 5.038, 5.766, 6.61, 7.534, 8.489, 9.252, 9.6,
00475 9.161, 7.958, 6.512, 5.259, 4.317, 3.547, 2.789, 2.13, 1.601,
00476 1.205, 0.9321, 0.7532, 0.6464, 0.6173, 0.5896, 0.5782, 1.014, 1.096,
00477 1.226, 1.387, 1.111, 1.042, 0.9908, 0.9408, 0.8311},
00478 {4.621, 4.534, 4.984, 5.693, 6.545, 7.49, 8.444, 9.177, 9.531,
00479 9.117, 7.928, 6.533, 5.27, 4.271, 3.431, 2.575, 1.902, 1.42,
00480 1.11, 0.9004, 0.7658, 0.6955, 0.7676, 0.9088, 0.8989, 1.028, 1.221,
00481 1.455, 1.583, 1.375, 1.376, 1.498, 1.744, 1.925},
00482 {4.514, 4.41, 4.837, 5.545, 6.416, 7.38, 8.287, 9.05, 9.416,
00483 9.022, 7.903, 6.496, 5.175, 4.111, 3.232, 2.38, 1.76, 1.335,
00484 1.068, 0.9227, 0.8515, 0.8511, 0.9534, 1.091, 0.4909, 0.9377, 1.241,
00485 1.592, 1.739, 1.478, 1.473, 1.597, 1.893, 2.117},
00486 {4.407, 4.264, 4.61, 5.263, 6.095, 7.046, 8.005, 8.805, 9.201,
00487 8.823, 7.705, 6.299, 4.964, 3.891, 3.046, 2.214, 1.641, 1.261,
00488 1.027, 0.922, 0.8759, 0.8893, 0.9782, 0.3707, 0.4349, 0.976, 1.523,
00489 2.021, 1.906, 1.524, 1.486, 1.53, 1.741, 1.869},
00490 {4.156, 4.007, 4.37, 4.987, 5.777, 6.719, 7.728, 8.578, 8.97,
00491 8.552, 7.409, 6.027, 4.731, 3.684, 2.814, 2.029, 1.501, 1.159,
00492 0.9542, 0.8666, 0.8191, 0.8371, 0.9704, 0.3324, 0.5634, 0.9279, 1.512,
00493 2.042, 1.951, 1.566, 1.535, 1.567, 1.739, 1.822},
00494 {3.98, 3.883, 4.232, 4.841, 5.594, 6.502, 7.497, 8.296, 8.631,
00495 8.161, 7.043, 5.703, 4.51, 3.548, 2.717, 1.951, 1.435, 1.107,
00496 0.9188, 0.8286, 0.772, 0.7564, 0.8263, 0.3026, 0.3854, 0.8743, 1.452,
00497 2.024, 2.012, 1.764, 1.85, 2.125, 2.621, 2.9},
00498 {3.877, 3.811, 4.167, 4.733, 5.462, 6.324, 7.144, 7.862, 8.14,
00499 7.695, 6.613, 5.393, 4.299, 3.42, 2.653, 1.947, 1.44, 1.1,
00500 0.8953, 0.7907, 0.7218, 0.6881, 0.7281, 0.8846, 0.3802, 0.6853, 1.025,
00501 1.593, 2.088, 1.846, 1.866, 2.108, 2.548, 2.737},
00502 {3.636, 3.634, 4.034, 4.637, 5.354, 6.148, 6.899, 7.495, 7.657,
00503 7.138, 6.107, 4.974, 3.983, 3.21, 2.558, 1.932, 1.439, 1.062,
00504 0.8104, 0.6632, 0.5721, 0.517, 0.5182, 0.501, 0.3937, 0.5565, 0.7675,
00505 0.9923, 1.455, 1.554, 1.567, 1.701, 2.077, 2.257},
00506 {3.366, 3.446, 3.885, 4.473, 5.16, 5.9, 6.557, 6.973, 6.973,
00507 6.379, 5.387, 4.385, 3.561, 2.941, 2.428, 1.848, 1.349, 0.9512,
00508 0.6956, 0.5432, 0.4437, 0.3733, 0.3297, 0.2972, 0.2518, 0.4613, 0.5078,
00509 0.6579, 0.8696, 0.9827, 1.028, 1.114, 1.275, 1.315},
00510 {2.975, 3.121, 3.567, 4.146, 4.797, 5.466, 6.003, 6.279, 6.087,
00511 5.372, 4.442, 3.614, 2.986, 2.521, 2.082, 1.532, 1.066, 0.7304,
00512 0.5316, 0.4123, 0.3314, 0.274, 0.2365, 0.2061, 0.1755, 0.1348, 0.2217,
00513 0.2854, 0.387, 0.4315, 0.4301, 0.4585, 0.5195, 0.5349},
00514 {2.434, 2.632, 3.054, 3.577, 4.156, 4.725, 5.1, 5.109, 4.7,
00515 3.958, 3.191, 2.588, 2.182, 1.866, 1.491, 1.034, 0.6936, 0.4774,
00516 0.3551, 0.2749, 0.2203, 0.1829, 0.1544, 0.1325, 0.1103, 0.07716, 0.08626,
00517 0.1037, 0.1455, 0.1418, 0.1351, 0.1339, 0.1407, 0.143},
00518 {1.798, 2.004, 2.333, 2.746, 3.164, 3.49, 3.572, 3.315, 2.833,
00519 2.269, 1.797, 1.485, 1.222, 0.9595, 0.6972, 0.4717, 0.3313, 0.2315,
00520 0.1727, 0.1407, 0.1204, 0.1116, 0.08791, 0.07567, 0.07432, 0.04138,
00521 0.02942,
00522 0.02407, 0.02969, 0.02808, 0.0261, 0.02364, 0.02079, 0.01623},
00523 {1.01, 1.1, 1.194, 1.284, 1.32, 1.278, 1.136, 0.9619, 0.815,
00524 0.7226, 0.6136, 0.472, 0.3124, 0.2056, 0.1448, 0.1041, 0.07233, 0.05587,
00525 0.05634, 0.04788, 0.03971, 0.02542, 0.01735, 0.01522, 0.01475, 0.01076,
00526 0.005235,
00527 0.003546, 0.003113, 0.003146, 0.002902, 0.002635, 0.001966, 0.001156},
00528 {0.04181, 0.03949, 0.03577, 0.03191, 0.02839, 0.02391, 0.01796, 0.01251,

```

```

00529      0.008422,
00530      0.005359, 0.003257, 0.001769, 0.001092, 0.0008405, 0.001744, 0.0003061,
00531      0.0002956, 0.000334,
00532      0.0004223, 0.0004062, 0.0003886, 0.0003418, 0.0003588, 0.0005709,
00533      0.001368, 0.0001472, 0.0003282,
00534      2.039e-06, 9.891e-13, 1.597e-13, 9.622e-14, 1.167e-13, 1.706e-13,
00535      3.59e-13}},
00536      {{3.361, 3.811, 4.35, 4.813, 5.132, 5.252, 5.019, 4.545, 3.913,
00537      3.25, 2.739, 2.417, 2.071, 1.582, 1.059, 0.6248, 0.3911, 0.2807,
00538      0.2095, 0.1539, 0.118, 0.0953, 0.08084, 0.07456, 0.07567, 0.07108, 0.1196,
00539      0.08475, 0.0839, 0.05724, 0.04673, 0.0412, 0.03626, 0.03134}},
00540      {3.193, 3.459, 3.973, 4.578, 5.172, 5.677, 5.945, 5.913, 5.487,
00541      4.686, 3.838, 3.217, 2.773, 2.409, 1.9, 1.33, 0.899, 0.64,
00542      0.4734, 0.355, 0.277, 0.2245, 0.1928, 0.1794, 0.1806, 0.1934, 0.3268,
00543      0.2217, 0.1969, 0.1375, 0.1206, 0.103, 0.08729, 0.07332}},
00544      {3.456, 3.617, 4.127, 4.778, 5.448, 6.083, 6.59, 6.858, 6.716,
00545      6.004, 4.975, 4.092, 3.442, 2.986, 2.474, 1.87, 1.328, 0.9231,
00546      0.6642, 0.5044, 0.4019, 0.336, 0.3049, 0.2996, 0.3227, 0.3571, 0.4295,
00547      0.3789, 0.3663, 0.3286, 0.3128, 0.3015, 0.302, 0.2968}},
00548      {3.743, 3.817, 4.311, 4.948, 5.666, 6.417, 7.077, 7.528, 7.551,
00549      6.893, 5.766, 4.721, 3.913, 3.326, 2.788, 2.144, 1.555, 1.093,
00550      0.7905, 0.6101, 0.4943, 0.4214, 0.3965, 0.3958, 0.4205, 0.4093, 0.5808,
00551      0.5876, 0.6359, 0.5956, 0.5731, 0.5485, 0.512, 0.4733}},
00552      {4.012, 4.012, 4.455, 5.113, 5.866, 6.661, 7.41, 7.98, 8.111,
00553      7.509, 6.364, 5.223, 4.308, 3.602, 2.951, 2.23, 1.625, 1.168,
00554      0.8748, 0.6916, 0.5758, 0.5122, 0.5053, 0.444, 0.4277, 0.7575, 0.7756,
00555      0.8848, 1.034, 0.8886, 0.8352, 0.7965, 0.7562, 0.68}},
00556      {4.136, 4.113, 4.549, 5.187, 5.93, 6.754, 7.592, 8.298, 8.546,
00557      8.022, 6.882, 5.668, 4.625, 3.778, 3.016, 2.236, 1.625, 1.193,
00558      0.9253, 0.756, 0.6523, 0.598, 0.6186, 0.6287, 0.4804, 0.8007, 0.9732,
00559      1.175, 1.329, 1.194, 1.193, 1.288, 1.47, 1.603}},
00560      {4.185, 4.119, 4.528, 5.186, 5.95, 6.801, 7.678, 8.404, 8.709,
00561      8.272, 7.179, 5.922, 4.767, 3.814, 2.996, 2.187, 1.599, 1.203,
00562      0.9596, 0.8307, 0.7617, 0.7489, 0.822, 0.9684, 0.4374, 0.7664, 1.044,
00563      1.36, 1.525, 1.331, 1.334, 1.447, 1.685, 1.862}},
00564      {4.213, 4.111, 4.507, 5.117, 5.882, 6.773, 7.67, 8.464, 8.829,
00565      8.438, 7.378, 6.088, 4.84, 3.802, 2.915, 2.114, 1.561, 1.195,
00566      0.9731, 0.8773, 0.8296, 0.8295, 0.9193, 1.114, 0.3986, 0.8793, 1.394,
00567      1.884, 1.786, 1.437, 1.403, 1.446, 1.64, 1.752}},
00568      {4.216, 4.092, 4.458, 5.04, 5.78, 6.67, 7.617, 8.47, 8.883,
00569      8.531, 7.504, 6.163, 4.862, 3.792, 2.891, 2.063, 1.519, 1.174,
00570      0.979, 0.8935, 0.8377, 0.8354, 0.9357, 0.3235, 0.4057, 0.9346, 1.53,
00571      2.078, 1.951, 1.555, 1.525, 1.557, 1.737, 1.825}},
00572      {4.168, 4.054, 4.417, 5.011, 5.76, 6.655, 7.601, 8.453, 8.882,
00573      8.539, 7.487, 6.143, 4.853, 3.809, 2.92, 2.088, 1.531, 1.181,
00574      0.9861, 0.8955, 0.8297, 0.8116, 0.8945, 1.137, 0.4003, 0.9318, 1.557,
00575      2.163, 2.095, 1.82, 1.931, 2.246, 2.823, 3.14}},
00576      {4.15, 4.078, 4.457, 5.044, 5.791, 6.692, 7.602, 8.343, 8.697,
00577      8.324, 7.282, 5.969, 4.743, 3.764, 2.926, 2.138, 1.572, 1.197,
00578      0.9808, 0.871, 0.7977, 0.764, 0.7663, 0.8665, 0.3969, 0.7472, 1.125,
00579      1.78, 2.277, 1.98, 2.036, 2.365, 2.936, 3.186}},
00580      {4.028, 4.006, 4.421, 5.039, 5.793, 6.657, 7.515, 8.216, 8.5,
00581      8.071, 7.01, 5.768, 4.638, 3.752, 3.021, 2.305, 1.714, 1.261,
00582      0.9646, 0.7989, 0.6935, 0.6314, 0.6426, 0.5914, 0.4424, 0.6827, 0.917,
00583      1.27, 1.864, 1.895, 1.922, 2.172, 2.772, 3.141}},
00584      {3.825, 3.864, 4.324, 4.97, 5.727, 6.565, 7.363, 7.978, 8.162,
00585      7.628, 6.542, 5.369, 4.372, 3.601, 2.977, 2.292, 1.697, 1.211,
00586      0.8808, 0.6852, 0.562, 0.4717, 0.4094, 0.366, 0.3093, 0.5639, 0.649,
00587      0.8282, 1.144, 1.27, 1.346, 1.522, 1.831, 1.939}},
00588      {3.493, 3.611, 4.102, 4.736, 5.491, 6.333, 7.098, 7.615, 7.666,
00589      7.006, 5.88, 4.791, 3.918, 3.283, 2.733, 2.099, 1.522, 1.066,
00590      0.7595, 0.5845, 0.4736, 0.3918, 0.337, 0.2962, 0.2564, 0.2024, 0.3657,
00591      0.4744, 0.6366, 0.76, 0.8059, 0.9015, 1.065, 1.126}},
00592      {3.167, 3.353, 3.857, 4.499, 5.225, 6.002, 6.699, 7.106, 6.98,
00593      6.192, 5.094, 4.125, 3.4, 2.87, 2.382, 1.774, 1.246, 0.8549,
00594      0.6089, 0.4666, 0.3762, 0.3082, 0.2643, 0.2283, 0.1942, 0.142, 0.1945,
00595      0.2512, 0.3593, 0.4048, 0.4081, 0.4206, 0.4616, 0.4738}},
00596      {2.839, 3.07, 3.565, 4.17, 4.846, 5.529, 6.041, 6.194, 5.833,
00597      5.009, 4.027, 3.239, 2.656, 2.227, 1.794, 1.282, 0.9003, 0.6219,
00598      0.4405, 0.3353, 0.2672, 0.2205, 0.1979, 0.1911, 0.1846, 0.1044, 0.09845,
00599      0.1069, 0.1547, 0.1611, 0.156, 0.1466, 0.1369, 0.1212}},
00600      {2.471, 2.75, 3.188, 3.712, 4.27, 4.77, 4.987, 4.802, 4.251,
00601      3.473, 2.731, 2.217, 1.83, 1.498, 1.137, 0.7594, 0.5383, 0.3859,
00602      0.2842, 0.2242, 0.1817, 0.1654, 0.1508, 0.1422, 0.1274, 0.07194, 0.05548,
00603      0.04058, 0.05226, 0.05347, 0.04942, 0.04415, 0.03177, 0.01923}},
00604      {2.192, 2.463, 2.735, 2.958, 3.106, 3.134, 2.927, 2.561, 2.149,
00605      1.825, 1.627, 1.429, 1.102, 0.766, 0.4943, 0.3166, 0.2184, 0.1471,
00606      0.1087, 0.08747, 0.07338, 0.06265, 0.05554, 0.05001, 0.04253, 0.03353,
00607      0.01781,
00608      0.01332, 0.01371, 0.01531, 0.01438, 0.01296, 0.00982, 0.006101}},
00609      {{0.2905, 0.282, 0.2639, 0.2366, 0.2078, 0.1879, 0.1644, 0.1407, 0.1163,
00610      0.08898, 0.06233, 0.04005, 0.02438, 0.01319, 0.007679, 0.005681, 0.004482,
00611      0.004329,
00612      0.004242, 0.004414, 0.004655, 0.004515, 0.003998, 0.00342, 0.003203,
00613      0.001672, 0.00135,
00614      0.001376, 0.0004656, 0.0003538, 0.000242, 0.0001647, 0.0001078,
00615      5.702e-05}},

```

```
00616 {1.5, 1.685, 1.901, 2.105, 2.216, 2.203, 2.048, 1.855, 1.609,
00617 1.357, 1.162, 1.012, 0.8042, 0.573, 0.372, 0.223, 0.144, 0.1095,
00618 0.08876, 0.07152, 0.05912, 0.05031, 0.04379, 0.03983, 0.03757, 0.03481,
00619 0.04463,
00620 0.02747, 0.02318, 0.01802, 0.0155, 0.01347, 0.01136, 0.009558},
00621 {2.228, 2.464, 2.889, 3.347, 3.773, 4.098, 4.191, 4.074, 3.7,
00622 3.132, 2.591, 2.217, 1.929, 1.594, 1.177, 0.7799, 0.5288, 0.3786,
00623 0.2805, 0.2144, 0.1695, 0.1403, 0.1236, 0.1157, 0.1134, 0.1148, 0.1127,
00624 0.1059, 0.0982, 0.08934, 0.08645, 0.08666, 0.09037, 0.09339},
00625 {2.761, 2.942, 3.406, 3.981, 4.592, 5.177, 5.591, 5.699, 5.391,
00626 4.682, 3.883, 3.267, 2.804, 2.378, 1.889, 1.347, 0.9255, 0.6382,
00627 0.4661, 0.361, 0.2904, 0.2459, 0.2237, 0.2143, 0.2142, 0.1959, 0.2522,
00628 0.2552, 0.2714, 0.274, 0.2676, 0.2624, 0.2589, 0.2436},
00629 {3.207, 3.317, 3.769, 4.373, 5.05, 5.759, 6.356, 6.677, 6.533,
00630 5.82, 4.861, 4.056, 3.414, 2.865, 2.322, 1.708, 1.204, 0.8386,
00631 0.6195, 0.4861, 0.3997, 0.3459, 0.32, 0.2881, 0.251, 0.4026, 0.4166,
00632 0.4798, 0.5723, 0.5711, 0.5498, 0.5348, 0.5181, 0.4851},
00633 {3.509, 3.549, 3.992, 4.607, 5.301, 6.054, 6.784, 7.302, 7.372,
00634 6.753, 5.706, 4.725, 3.909, 3.223, 2.58, 1.894, 1.356, 0.9716,
00635 0.7397, 0.5944, 0.4998, 0.4401, 0.4193, 0.3649, 0.3204, 0.5317, 0.5992,
00636 0.7541, 0.9257, 0.9193, 0.9167, 0.976, 1.081, 1.167},
00637 {3.748, 3.717, 4.128, 4.736, 5.472, 6.31, 7.086, 7.725, 7.909,
00638 7.387, 6.323, 5.206, 4.217, 3.376, 2.638, 1.912, 1.382, 1.026,
00639 0.8072, 0.6849, 0.6123, 0.5755, 0.6121, 0.6279, 0.3644, 0.5863, 0.823,
00640 1.068, 1.241, 1.134, 1.134, 1.222, 1.397, 1.55},
00641 {3.966, 3.863, 4.239, 4.826, 5.583, 6.465, 7.317, 8.033, 8.34,
00642 7.916, 6.843, 5.63, 4.49, 3.524, 2.698, 1.933, 1.409, 1.074,
00643 0.8747, 0.7756, 0.7211, 0.7103, 0.7513, 0.3147, 0.3627, 0.7582, 1.206,
00644 1.64, 1.639, 1.321, 1.285, 1.334, 1.496, 1.611},
00645 {4.184, 4.066, 4.42, 4.981, 5.712, 6.597, 7.502, 8.305, 8.719,
00646 8.382, 7.314, 6.02, 4.76, 3.702, 2.787, 1.993, 1.461, 1.13,
00647 0.9379, 0.8456, 0.7865, 0.7693, 0.835, 0.3132, 0.4006, 0.9192, 1.486,
00648 1.956, 1.836, 1.485, 1.462, 1.502, 1.65, 1.74},
00649 {4.304, 4.184, 4.566, 5.178, 5.931, 6.823, 7.761, 8.572, 9.013,
00650 8.72, 7.64, 6.283, 4.975, 3.89, 2.959, 2.123, 1.552, 1.194,
00651 0.9949, 0.8938, 0.8268, 0.814, 0.8837, 0.3629, 0.449, 1.027, 1.698,
00652 2.339, 2.232, 1.709, 1.64, 1.636, 1.794, 1.865},
00653 {4.354, 4.279, 4.704, 5.365, 6.153, 7.049, 7.975, 8.793, 9.218,
00654 8.889, 7.805, 6.42, 5.093, 4.013, 3.112, 2.283, 1.671, 1.263,
00655 1.032, 0.9124, 0.8364, 0.8112, 0.8709, 0.9784, 0.5043, 0.9917, 1.587,
00656 2.354, 2.467, 1.85, 1.738, 1.724, 1.873, 1.918},
00657 {4.35, 4.331, 4.785, 5.473, 6.289, 7.199, 8.133, 8.898, 9.288,
00658 8.907, 7.796, 6.411, 5.112, 4.065, 3.212, 2.421, 1.788, 1.331,
00659 1.04, 0.8749, 0.769, 0.7242, 0.7799, 0.8189, 0.5724, 0.86, 1.151,
00660 1.587, 1.946, 1.722, 1.71, 1.845, 2.146, 2.307},
00661 {4.268, 4.264, 4.774, 5.45, 6.298, 7.255, 8.148, 8.907, 9.242,
00662 8.793, 7.612, 6.244, 5.029, 4.124, 3.411, 2.65, 1.98, 1.442,
00663 1.074, 0.8493, 0.7077, 0.6157, 0.5826, 0.5377, 0.4848, 0.7245, 0.8927,
00664 1.153, 1.565, 1.551, 1.638, 1.846, 2.153, 2.304},
00665 {4.119, 4.16, 4.652, 5.372, 6.224, 7.158, 8.06, 8.773, 9.034,
00666 8.507, 7.307, 5.989, 4.866, 4.046, 3.387, 2.637, 1.959, 1.402,
00667 1.01, 0.7722, 0.6266, 0.527, 0.462, 0.4124, 0.3452, 0.6418, 0.572,
00668 0.7352, 1.009, 1.245, 1.364, 1.556, 1.828, 1.914},
00669 {3.932, 4.041, 4.572, 5.285, 6.144, 7.099, 7.985, 8.671, 8.849,
00670 8.203, 6.936, 5.657, 4.621, 3.877, 3.266, 2.526, 1.845, 1.302,
00671 0.9214, 0.6906, 0.5486, 0.4553, 0.4002, 0.3533, 0.3033, 0.24, 0.4062,
00672 0.5084, 0.7133, 0.9299, 0.9714, 1.073, 1.204, 1.211},
00673 {3.882, 4.074, 4.658, 5.414, 6.269, 7.182, 8.05, 8.635, 8.639,
00674 7.828, 6.529, 5.303, 4.339, 3.626, 3.001, 2.262, 1.617, 1.128,
00675 0.7869, 0.574, 0.4488, 0.3713, 0.3216, 0.2794, 0.2403, 0.1962, 0.2638,
00676 0.3018, 0.4483, 0.5618, 0.5717, 0.5812, 0.5993, 0.5956},
00677 {4.195, 4.559, 5.287, 6.13, 7.024, 7.911, 8.623, 8.913, 8.545,
00678 7.44, 6.059, 4.92, 4.063, 3.391, 2.747, 1.953, 1.327, 0.9036,
00679 0.6235, 0.4566, 0.3534, 0.2935, 0.2547, 0.2226, 0.1871, 0.1494, 0.1959,
00680 0.1797, 0.2736, 0.3214, 0.3054, 0.2803, 0.2443, 0.1504},
00681 {4.644, 5.211, 6.192, 7.373, 8.534, 9.582, 10.35, 10.37, 9.49,
00682 7.907, 6.299, 5.088, 4.224, 3.492, 2.701, 1.795, 1.184, 0.8001,
00683 0.5348, 0.3789, 0.2861, 0.2435, 0.2055, 0.1754, 0.1605, 0.1076, 0.1644,
00684 0.09069, 0.1624, 0.2356, 0.223, 0.2001, 0.164, 0.09807},
00685 {{0.0001362, 0.0001753, 0.0001753, 0.0001881, 0.0002233, 0.0002056,
00686 0.0001083, 1.505e-05, 1.073e-05,
00687 1.22e-05, 7.475e-06, 3.518e-06, 1.292e-06, 3.812e-07, 8.611e-08,
00688 3.148e-09, 1.29e-09, 2.111e-08,
00689 7.591e-07, 1.125e-06, 4.905e-06, 1.473e-05, 3.597e-05, 9.35e-05,
00690 0.0002623, 0.0001066, 0.0002064,
00691 0.0001924, 6.532e-10, 3.205e-10, 3.838e-10, 3.541e-10, 3.331e-10,
00692 2.774e-10},
00693 {0.1865, 0.1757, 0.1596, 0.1454, 0.1343, 0.125, 0.1033, 0.08324, 0.06202,
00694 0.04341, 0.0295, 0.0187, 0.01129, 0.007115, 0.004932, 0.003824, 0.002733,
00695 0.003019,
00696 0.004022, 0.004428, 0.004414, 0.003826, 0.003009, 0.002117, 0.001375,
00697 0.0009066, 0.001031,
00698 0.0004673, 0.0002194, 0.0001683, 0.0001154, 6.607e-05, 3.128e-05,
00699 1.601e-05},
00700 {1.238, 1.384, 1.576, 1.788, 1.95, 1.993, 1.845, 1.593, 1.329,
00701 1.098, 0.9387, 0.8174, 0.6531, 0.4579, 0.2899, 0.1818, 0.1253, 0.09461,
00702 0.0741, 0.05946, 0.04989, 0.04353, 0.03862, 0.03469, 0.03181, 0.02862,
```

00703 0.02499,
 00704 0.02062, 0.0178, 0.01681, 0.0162, 0.01679, 0.01921, 0.02192},
 00705 {1.92, 2.106, 2.478, 2.931, 3.421, 3.86, 4.043, 3.866, 3.41,
 00706 2.84, 2.372, 2.077, 1.825, 1.472, 1.07, 0.7023, 0.4688, 0.3296,
 00707 0.2453, 0.1889, 0.15, 0.1246, 0.1094, 0.09788, 0.09079, 0.08267, 0.08977,
 00708 0.09461, 0.09867, 0.09977, 0.09675, 0.09757, 0.1011, 0.1004},
 00709 {2.505, 2.661, 3.095, 3.659, 4.295, 4.928, 5.368, 5.413, 5.018,
 00710 4.291, 3.582, 3.052, 2.636, 2.193, 1.714, 1.199, 0.8121, 0.5616,
 00711 0.4193, 0.3284, 0.2656, 0.2245, 0.2009, 0.1819, 0.1615, 0.2142, 0.2142,
 00712 0.2523, 0.3029, 0.3253, 0.3248, 0.323, 0.3186, 0.3005},
 00713 {3.053, 3.153, 3.582, 4.16, 4.823, 5.525, 6.111, 6.38, 6.176,
 00714 5.453, 4.575, 3.84, 3.219, 2.652, 2.09, 1.509, 1.055, 0.7478,
 00715 0.5678, 0.4529, 0.3746, 0.322, 0.2979, 0.2602, 0.284, 0.3515, 0.39,
 00716 0.4915, 0.6421, 0.6745, 0.684, 0.7208, 0.7815, 0.8179},
 00717 {3.422, 3.44, 3.875, 4.477, 5.191, 5.965, 6.661, 7.123, 7.132,
 00718 6.48, 5.463, 4.526, 3.718, 3.005, 2.344, 1.688, 1.206, 0.8837,
 00719 0.6856, 0.5649, 0.4863, 0.445, 0.4856, 0.5006, 0.3136, 0.4515, 0.6109,
 00720 0.7863, 0.9684, 0.937, 0.937, 0.9949, 1.106, 1.204},
 00721 {3.782, 3.708, 4.085, 4.684, 5.42, 6.264, 7.105, 7.757, 7.965,
 00722 7.421, 6.309, 5.169, 4.161, 3.312, 2.538, 1.814, 1.314, 0.9921,
 00723 0.7986, 0.6906, 0.6241, 0.6037, 0.6883, 0.3135, 0.3539, 0.6781, 1.044,
 00724 1.41, 1.465, 1.198, 1.162, 1.195, 1.315, 1.399},
 00725 {4.093, 3.956, 4.303, 4.881, 5.64, 6.546, 7.473, 8.261, 8.617,
 00726 8.211, 7.109, 5.806, 4.611, 3.616, 2.718, 1.923, 1.396, 1.077,
 00727 0.8999, 0.7993, 0.7318, 0.69, 0.7736, 0.3307, 0.5444, 0.8973, 1.424,
 00728 1.867, 1.76, 1.423, 1.397, 1.429, 1.558, 1.631},
 00729 {4.316, 4.179, 4.572, 5.214, 6.033, 6.982, 7.93, 8.754, 9.154,
 00730 8.809, 7.719, 6.283, 4.962, 3.898, 2.961, 2.106, 1.526, 1.178,
 00731 0.9911, 0.8826, 0.8091, 0.792, 0.8984, 0.4081, 0.6591, 1.079, 1.762,
 00732 2.394, 2.262, 1.722, 1.649, 1.64, 1.793, 1.856},
 00733 {4.422, 4.322, 4.787, 5.478, 6.345, 7.323, 8.254, 9.118, 9.564,
 00734 9.218, 8.075, 6.604, 5.203, 4.058, 3.117, 2.274, 1.677, 1.284,
 00735 1.048, 0.9151, 0.8297, 0.7966, 0.8935, 0.5312, 0.7531, 1.072, 1.715,
 00736 2.495, 2.546, 1.897, 1.782, 1.77, 1.93, 1.983},
 00737 {4.547, 4.469, 4.942, 5.688, 6.549, 7.488, 8.486, 9.332, 9.775,
 00738 9.417, 8.264, 6.785, 5.379, 4.249, 3.33, 2.505, 1.852, 1.39,
 00739 1.098, 0.9308, 0.8263, 0.7895, 0.8576, 1.037, 0.635, 0.9803, 1.371,
 00740 1.867, 2.133, 1.823, 1.82, 1.997, 2.372, 2.584},
 00741 {4.687, 4.634, 5.104, 5.806, 6.687, 7.693, 8.664, 9.479, 9.89,
 00742 9.543, 8.412, 6.918, 5.518, 4.426, 3.541, 2.711, 2.014, 1.493,
 00743 1.144, 0.9317, 0.797, 0.7099, 0.6575, 0.6067, 0.526, 0.9154, 1.118,
 00744 1.507, 1.832, 1.788, 1.887, 2.179, 2.647, 2.868},
 00745 {4.695, 4.698, 5.185, 5.955, 6.871, 7.893, 8.877, 9.693, 10.1,
 00746 9.723, 8.528, 6.981, 5.591, 4.567, 3.788, 2.967, 2.242, 1.655,
 00747 1.222, 0.9406, 0.7599, 0.6428, 0.57, 0.513, 0.4496, 0.9249, 0.8597,
 00748 1.064, 1.428, 1.661, 1.831, 2.151, 2.551, 2.772},
 00749 {4.817, 4.823, 5.364, 6.154, 7.109, 8.194, 9.225, 10.09, 10.53,
 00750 10.13, 8.82, 7.204, 5.79, 4.774, 3.99, 3.147, 2.374, 1.734,
 00751 1.249, 0.9272, 0.7213, 0.5919, 0.522, 0.4691, 0.4205, 0.3435, 0.704,
 00752 0.795, 1.078, 1.369, 1.506, 1.697, 1.921, 2.057},
 00753 {5.178, 5.275, 5.904, 6.794, 7.806, 8.901, 9.941, 10.76, 11.06,
 00754 10.41, 8.899, 7.194, 5.78, 4.767, 3.992, 3.132, 2.304, 1.622,
 00755 1.125, 0.8041, 0.6135, 0.4964, 0.4334, 0.388, 0.3494, 0.3124, 0.5608,
 00756 0.6202, 0.8549, 1.142, 1.185, 1.327, 1.469, 1.521},
 00757 {5.765, 5.982, 6.854, 7.999, 9.258, 10.55, 11.75, 12.58, 12.65,
 00758 11.51, 9.567, 7.638, 6.122, 5.016, 4.17, 3.208, 2.283, 1.528,
 00759 0.9814, 0.6677, 0.4925, 0.3884, 0.3284, 0.2876, 0.2571, 0.2207, 0.5107,
 00760 0.5167, 0.7005, 0.8149, 0.7878, 0.7521, 0.6899, 0.4697},
 00761 {5.862, 6.014, 6.876, 8.048, 9.368, 10.8, 12.34, 13.65, 13.83,
 00762 12.41, 10.19, 8.106, 6.468, 5.285, 4.422, 3.392, 2.324, 1.509,
 00763 0.9796, 0.6834, 0.5019, 0.387, 0.3295, 0.2804, 0.2425, 0.2084, 0.4694,
 00764 0.4497, 0.5949, 0.6504, 0.6155, 0.5696, 0.5077, 0.2744}},
 00765 {{6.617e-05, 8.467e-05, 8.509e-05, 9.824e-05, 0.0001317, 0.0001499,
 00766 0.0001104, 1.226e-05, 1.003e-05,
 00767 1.345e-05, 8.036e-06, 2.28e-06, 3.166e-07, 1.803e-08, 6.079e-10,
 00768 3.031e-10, 1.336e-09, 1.748e-08,
 00769 3.057e-07, 8.184e-07, 1.95e-06, 2.238e-06, 4.658e-06, 1.393e-05,
 00770 4.326e-05, 3.288e-05, 0.0001662,
 00771 8.012e-05, 6.48e-10, 3.371e-10, 4.509e-10, 4.052e-10, 3.677e-10,
 00772 2.996e-10},
 00773 {0.003936, 0.002158, 0.001688, 0.001555, 0.001396, 0.0008637, 0.0003091,
 00774 4.908e-05, 1.173e-05,
 00775 1.326e-05, 9.097e-06, 3.324e-06, 6.336e-07, 8.553e-08, 4.851e-09,
 00776 3.058e-09, 9.169e-09, 2.451e-07,
 00777 8.77e-07, 2.973e-07, 2.556e-07, 2.79e-07, 8.674e-07, 4.17e-06, 2.373e-05,
 00778 3.876e-05, 6.493e-05,
 00779 2.13e-05, 2.574e-09, 2.676e-10, 2.356e-10, 1.947e-10, 2.376e-10,
 00780 1.955e-10},
 00781 {0.8642, 0.927, 0.9782, 1.001, 0.9705, 0.885, 0.7521, 0.6223, 0.5357,
 00782 0.4614, 0.373, 0.2955, 0.2138, 0.1342, 0.07775, 0.04773, 0.0348, 0.02871,
 00783 0.02503, 0.02329, 0.02308, 0.02227, 0.01967, 0.01687, 0.01381, 0.008063,
 00784 0.006489,
 00785 0.005433, 0.004804, 0.00487, 0.004735, 0.004696, 0.004587, 0.003634},
 00786 {1.584, 1.731, 2.006, 2.368, 2.742, 3.047, 3.124, 2.894, 2.517,
 00787 2.087, 1.761, 1.588, 1.399, 1.091, 0.7549, 0.4844, 0.3282, 0.2357,
 00788 0.1737, 0.1299, 0.1001, 0.08162, 0.0699, 0.06056, 0.05333, 0.07014,
 00789 0.04552,


```

00790     0.04907, 0.04781, 0.04938, 0.04864, 0.04971, 0.05141, 0.05078},
00791     {2.256, 2.4, 2.786, 3.287, 3.854, 4.41, 4.773, 4.776, 4.396,
00792     3.755, 3.16, 2.747, 2.381, 1.94, 1.458, 0.9865, 0.6576, 0.4582,
00793     0.3447, 0.2672, 0.2121, 0.1768, 0.1538, 0.1358, 0.1219, 0.1466, 0.1371,
00794     0.161, 0.1896, 0.2116, 0.2133, 0.2153, 0.2145, 0.2042},
00795     {2.844, 2.913, 3.296, 3.841, 4.473, 5.127, 5.672, 5.933, 5.724,
00796     5.005, 4.187, 3.549, 3.006, 2.486, 1.928, 1.368, 0.9363, 0.6524,
00797     0.4946, 0.3921, 0.3189, 0.2696, 0.2503, 0.2181, 0.2227, 0.2946, 0.297,
00798     0.3704, 0.4834, 0.5473, 0.5565, 0.581, 0.6189, 0.6392},
00799     {3.309, 3.337, 3.765, 4.35, 5.025, 5.756, 6.438, 6.908, 6.947,
00800     6.3, 5.252, 4.33, 3.576, 2.912, 2.258, 1.618, 1.15, 0.8271,
00801     0.6351, 0.5129, 0.4287, 0.3846, 0.4351, 0.4889, 0.3229, 0.4124, 0.4839,
00802     0.6285, 0.8468, 0.9091, 0.9059, 0.9413, 1.007, 1.074},
00803     {3.725, 3.662, 4.042, 4.647, 5.381, 6.204, 7.019, 7.636, 7.841,
00804     7.318, 6.177, 5.04, 4.09, 3.299, 2.545, 1.822, 1.303, 0.967,
00805     0.7683, 0.6506, 0.5816, 0.5712, 0.667, 0.3774, 0.3833, 0.5497, 0.7343,
00806     0.9899, 1.25, 1.213, 1.147, 1.144, 1.27, 1.389},
00807     {4.074, 3.916, 4.273, 4.851, 5.624, 6.564, 7.525, 8.296, 8.632,
00808     8.184, 7.013, 5.711, 4.593, 3.691, 2.813, 2.003, 1.445, 1.097,
00809     0.8982, 0.7815, 0.7103, 0.6929, 0.7841, 0.3697, 0.42, 0.8027, 1.255,
00810     1.716, 1.727, 1.411, 1.379, 1.416, 1.548, 1.618},
00811     {4.378, 4.233, 4.657, 5.297, 6.121, 7.098, 8.11, 8.937, 9.31,
00812     8.877, 7.696, 6.289, 5.029, 3.997, 3.066, 2.189, 1.589, 1.209,
00813     0.9953, 0.8642, 0.7883, 0.7739, 0.9011, 0.4115, 0.6312, 0.9904, 1.555,
00814     2.033, 1.915, 1.547, 1.523, 1.55, 1.675, 1.75},
00815     {4.574, 4.464, 4.871, 5.569, 6.471, 7.487, 8.486, 9.356, 9.788,
00816     9.387, 8.196, 6.717, 5.357, 4.245, 3.29, 2.387, 1.738, 1.316,
00817     1.064, 0.9101, 0.8224, 0.8068, 0.9022, 0.478, 0.7124, 1.093, 1.75,
00818     2.355, 2.233, 1.721, 1.663, 1.652, 1.836, 1.906},
00819     {4.771, 4.625, 5.041, 5.796, 6.698, 7.684, 8.72, 9.611, 10.08,
00820     9.736, 8.573, 7.013, 5.571, 4.447, 3.501, 2.606, 1.915, 1.439,
00821     1.139, 0.956, 0.8517, 0.8306, 0.9424, 1.177, 0.5869, 1.036, 1.44,
00822     1.878, 2.039, 1.705, 1.737, 1.923, 2.385, 2.641},
00823     {5.006, 4.866, 5.324, 6.09, 7.006, 8.036, 9.084, 9.893, 10.33,
00824     10.02, 8.839, 7.228, 5.736, 4.611, 3.71, 2.8, 2.063, 1.532,
00825     1.183, 0.9673, 0.8363, 0.7853, 0.8359, 0.8923, 0.569, 1.039, 1.365,
00826     1.742, 1.904, 1.758, 1.875, 2.251, 2.843, 3.224},
00827     {5.099, 4.961, 5.447, 6.211, 7.152, 8.22, 9.322, 10.2, 10.72,
00828     10.48, 9.248, 7.559, 5.993, 4.842, 3.975, 3.116, 2.35, 1.736,
00829     1.297, 1.003, 0.8134, 0.702, 0.6582, 0.6213, 0.5842, 0.5417, 1.158,
00830     1.4, 1.637, 1.863, 2.063, 2.497, 3.122, 3.523},
00831     {5.329, 5.217, 5.717, 6.57, 7.585, 8.724, 9.852, 10.78, 11.29,
00832     10.97, 9.671, 7.914, 6.3, 5.116, 4.263, 3.389, 2.58, 1.892,
00833     1.383, 1.036, 0.8131, 0.6799, 0.6149, 0.5698, 0.5283, 0.4671, 1.093,
00834     1.188, 1.482, 1.733, 1.908, 2.209, 2.612, 2.814},
00835     {6.071, 6.113, 6.75, 7.703, 8.826, 10.05, 11.16, 11.96, 12.25,
00836     11.61, 10.02, 8.102, 6.438, 5.24, 4.366, 3.448, 2.553, 1.795,
00837     1.245, 0.8829, 0.6737, 0.5561, 0.4996, 0.4579, 0.4257, 0.4016, 0.968,
00838     1.042, 1.284, 1.504, 1.639, 1.849, 2.098, 2.255},
00839     {6.417, 6.432, 7.23, 8.422, 9.846, 11.4, 12.82, 13.78, 14,
00840     12.98, 10.94, 8.756, 6.951, 5.69, 4.77, 3.716, 2.672, 1.818,
00841     1.224, 0.8517, 0.6496, 0.5355, 0.4766, 0.4254, 0.389, 0.351, 0.9729,
00842     0.9755, 1.108, 1.026, 0.9967, 0.9687, 0.9329, 0.8183},
00843     {6.462, 6.445, 7.221, 8.396, 9.796, 11.38, 12.96, 14.19, 14.59,
00844     13.52, 11.33, 9.046, 7.189, 5.92, 5.015, 3.949, 2.77, 1.841,
00845     1.239, 0.86, 0.6441, 0.5179, 0.446, 0.42, 0.3818, 0.3068, 0.9267,
00846     0.8849, 0.9828, 0.8458, 0.7544, 0.6963, 0.6233, 0.5358}},
00847     {{3.24e-05, 4.086e-05, 4.221e-05, 5.381e-05, 8.808e-05, 0.0001261,
00848     0.0001263, 2.932e-05, 2.057e-05,
00849     4.321e-05, 3.43e-05, 1.213e-05, 1.862e-06, 1.067e-07, 2.983e-09,
00850     2.578e-08, 2.254e-09, 1.504e-08,
00851     1.599e-07, 1.523e-07, 2.033e-07, 5.271e-07, 1.417e-06, 4.518e-06,
00852     1.358e-05, 9.266e-06, 0.0001569,
00853     3.355e-05, 7.653e-10, 3.889e-10, 5.082e-10, 4.498e-10, 4.072e-10,
00854     3.302e-10},
00855     {0.0688, 0.05469, 0.04502, 0.03604, 0.02861, 0.02128, 0.01453, 0.00924,
00856     0.00574,
00857     0.003447, 0.001888, 0.001147, 0.001154, 0.001548, 0.001193, 6.955e-05,
00858     0.0003258, 1.673e-05,
00859     1.393e-05, 1.849e-05, 2.841e-05, 4.045e-05, 5.864e-05, 4.426e-05,
00860     1.722e-05, 3.186e-05, 5.582e-05,
00861     6.823e-06, 3.559e-09, 3.282e-10, 2.847e-10, 2.3e-10, 2.728e-10,
00862     2.237e-10},
00863     {1.056, 1.158, 1.28, 1.377, 1.429, 1.395, 1.242, 1.043, 0.8709,
00864     0.7435, 0.6183, 0.4824, 0.3665, 0.2668, 0.1749, 0.098, 0.09722, 0.08524,
00865     0.05545, 0.04501, 0.04476, 0.04491, 0.04231, 0.03868, 0.03336, 0.01108,
00866     0.01596,
00867     0.009172, 0.008499, 0.008763, 0.008672, 0.008872, 0.008926, 0.007552},
00868     {1.798, 1.952, 2.239, 2.589, 2.958, 3.261, 3.328, 3.12, 2.741,
00869     2.293, 1.917, 1.686, 1.53, 1.28, 0.9362, 0.6052, 0.4093, 0.294,
00870     0.2166, 0.1609, 0.1206, 0.09532, 0.08094, 0.07007, 0.06252, 0.1111,
00871     0.04996,
00872     0.0572, 0.0583, 0.06137, 0.06077, 0.0616, 0.0644, 0.06454},
00873     {2.42, 2.566, 2.952, 3.45, 3.987, 4.499, 4.853, 4.878, 4.55,
00874     3.918, 3.264, 2.802, 2.467, 2.094, 1.632, 1.127, 0.7523, 0.5168,
00875     0.386, 0.2982, 0.2346, 0.1928, 0.1694, 0.1534, 0.1384, 0.1672, 0.1434,
00876     0.1726, 0.2115, 0.2414, 0.2445, 0.2487, 0.2496, 0.2381},

```

00877 {3.026, 3.096, 3.501, 4.016, 4.608, 5.244, 5.786, 6.088, 5.975,
 00878 5.326, 4.429, 3.701, 3.142, 2.636, 2.071, 1.477, 1.015, 0.7056,
 00879 0.5315, 0.4215, 0.3413, 0.2869, 0.2681, 0.2507, 0.2616, 0.3004, 0.3042,
 00880 0.3879, 0.5203, 0.6045, 0.6196, 0.651, 0.6986, 0.7171},
 00881 {3.463, 3.475, 3.878, 4.441, 5.112, 5.838, 6.522, 7.015, 7.096,
 00882 6.52, 5.487, 4.496, 3.734, 3.105, 2.456, 1.776, 1.26, 0.9016,
 00883 0.6841, 0.5485, 0.454, 0.404, 0.4609, 0.5432, 0.376, 0.4702, 0.5045,
 00884 0.6555, 0.9157, 1.076, 1.113, 1.216, 1.372, 1.497},
 00885 {3.745, 3.679, 4.071, 4.666, 5.394, 6.22, 7.064, 7.725, 7.961,
 00886 7.448, 6.321, 5.161, 4.233, 3.472, 2.73, 1.96, 1.409, 1.04,
 00887 0.8213, 0.6863, 0.6068, 0.5872, 0.7122, 0.9507, 0.4302, 0.5845, 0.768,
 00888 1.036, 1.308, 1.372, 1.393, 1.57, 2.023, 2.387},
 00889 {4.046, 3.927, 4.286, 4.869, 5.645, 6.599, 7.567, 8.4, 8.74,
 00890 8.247, 7.036, 5.735, 4.638, 3.741, 2.905, 2.066, 1.493, 1.132,
 00891 0.9307, 0.7993, 0.7306, 0.7334, 0.8811, 1.197, 0.4369, 0.7932, 1.239,
 00892 1.716, 1.764, 1.639, 1.738, 2.042, 2.543, 2.814},
 00893 {4.303, 4.169, 4.566, 5.224, 6.055, 7.053, 8.124, 9.037, 9.471,
 00894 8.998, 7.714, 6.277, 4.981, 3.935, 3.018, 2.182, 1.599, 1.224,
 00895 1.01, 0.8634, 0.7916, 0.7955, 0.9234, 1.237, 0.6362, 0.9588, 1.513,
 00896 1.996, 1.902, 1.545, 1.53, 1.572, 1.712, 1.768},
 00897 {4.506, 4.381, 4.819, 5.529, 6.428, 7.462, 8.518, 9.453, 9.907,
 00898 9.467, 8.237, 6.703, 5.29, 4.149, 3.213, 2.346, 1.732, 1.324,
 00899 1.075, 0.9181, 0.8397, 0.8375, 0.9434, 0.5018, 0.5414, 1.083, 1.74,
 00900 2.347, 2.227, 1.721, 1.661, 1.648, 1.831, 1.912},
 00901 {4.687, 4.558, 4.978, 5.725, 6.658, 7.731, 8.789, 9.675, 10.13,
 00902 9.748, 8.526, 6.958, 5.498, 4.351, 3.431, 2.537, 1.87, 1.415,
 00903 1.127, 0.9521, 0.8587, 0.8423, 0.8885, 1.184, 0.5938, 1.064, 1.469,
 00904 1.877, 2.006, 1.669, 1.7, 1.879, 2.347, 2.626},
 00905 {4.936, 4.809, 5.237, 6.008, 6.937, 7.97, 9.016, 9.924, 10.41,
 00906 10.04, 8.758, 7.153, 5.667, 4.529, 3.633, 2.737, 2.011, 1.493,
 00907 1.161, 0.9601, 0.8439, 0.8078, 0.8976, 1.023, 0.6551, 1.239, 1.595,
 00908 1.9, 1.923, 1.721, 1.833, 2.227, 2.857, 3.283},
 00909 {5.075, 4.962, 5.428, 6.241, 7.2, 8.253, 9.328, 10.2, 10.65,
 00910 10.28, 8.983, 7.308, 5.772, 4.63, 3.752, 2.906, 2.151, 1.57,
 00911 1.188, 0.9525, 0.8028, 0.7168, 0.7021, 0.7201, 0.7339, 0.6036, 1.423,
 00912 1.574, 1.715, 1.841, 2.05, 2.525, 3.214, 3.663},
 00913 {5.33, 5.234, 5.731, 6.529, 7.518, 8.638, 9.699, 10.59, 11.06,
 00914 10.69, 9.363, 7.602, 6.028, 4.875, 4.029, 3.2, 2.426, 1.77,
 00915 1.305, 0.9937, 0.7977, 0.6831, 0.6247, 0.6058, 0.5859, 0.5171, 1.306,
 00916 1.309, 1.552, 1.652, 1.83, 2.179, 2.619, 2.844},
 00917 {5.866, 5.848, 6.455, 7.346, 8.396, 9.533, 10.53, 11.34, 11.68,
 00918 11.13, 9.619, 7.806, 6.227, 5.073, 4.235, 3.4, 2.554, 1.805,
 00919 1.264, 0.9221, 0.7291, 0.623, 0.5765, 0.5499, 0.5313, 0.5058, 1.13,
 00920 1.193, 1.384, 1.521, 1.658, 1.903, 2.204, 2.394},
 00921 {6.303, 6.398, 7.222, 8.351, 9.705, 11.15, 12.37, 13.21, 13.37,
 00922 12.4, 10.47, 8.393, 6.679, 5.472, 4.597, 3.653, 2.67, 1.827,
 00923 1.234, 0.8789, 0.697, 0.5977, 0.5427, 0.5065, 0.4765, 0.4416, 1.108,
 00924 1.122, 1.2, 1.026, 0.9995, 0.9762, 0.9406, 0.8155},
 00925 {6.374, 6.361, 7.22, 8.387, 9.771, 11.27, 12.69, 13.72, 14.07,
 00926 13.07, 10.88, 8.681, 6.947, 5.79, 4.913, 3.885, 2.773, 1.855,
 00927 1.244, 0.9142, 0.7256, 0.598, 0.5151, 0.5035, 0.4732, 0.3585, 1.059,
 00928 1.045, 1.089, 0.8483, 0.7697, 0.7077, 0.6288, 0.5436},
 00929 {{0.03789, 0.03408, 0.03134, 0.02846, 0.02395, 0.01876, 0.01296, 0.008683,
 00930 0.005595,
 00931 0.003327, 0.001899, 0.001016, 0.0006645, 0.001147, 0.000686, 0.0006987,
 00932 0.0006744, 0.0003903,
 00933 1.823e-05, 1.418e-05, 1.097e-05, 9.197e-06, 8.385e-06, 5.361e-06,
 00934 1.047e-05, 8.724e-06, 0.0001224,
 00935 1.608e-05, 7.761e-10, 4.005e-10, 5.216e-10, 4.653e-10, 4.282e-10,
 00936 3.472e-10},
 00937 {0.9685, 1.035, 1.103, 1.161, 1.165, 1.106, 0.9814, 0.8435, 0.7222,
 00938 0.6291, 0.5477, 0.4518, 0.3058, 0.2105, 0.1492, 0.08369, 0.1841, 0.1379,
 00939 0.1032, 0.06085, 0.03194, 0.02288, 0.0206, 0.02385, 0.0298, 0.0102,
 00940 0.002722,
 00941 0.00743, 0.007294, 0.006871, 0.006275, 0.005435, 0.004153, 0.002587},
 00942 {1.842, 2.002, 2.295, 2.618, 2.95, 3.205, 3.226, 2.966, 2.55,
 00943 2.078, 1.668, 1.355, 1.079, 0.8142, 0.7131, 0.5158, 0.3251, 0.2412,
 00944 0.1924, 0.1651, 0.1599, 0.164, 0.1325, 0.1215, 0.09591, 0.1117, 0.03993,
 00945 0.04814, 0.04778, 0.04729, 0.04627, 0.0467, 0.04597, 0.04172},
 00946 {2.489, 2.643, 3.032, 3.507, 4.035, 4.533, 4.829, 4.772, 4.375,
 00947 3.714, 3.06, 2.551, 2.207, 1.938, 1.628, 1.185, 0.8109, 0.5625,
 00948 0.4101, 0.3078, 0.2325, 0.1813, 0.1507, 0.1311, 0.1178, 0.1964, 0.1063,
 00949 0.1261, 0.1425, 0.1561, 0.1546, 0.1532, 0.1543, 0.1498},
 00950 {3.03, 3.156, 3.579, 4.144, 4.775, 5.42, 5.935, 6.129, 5.878,
 00951 5.138, 4.235, 3.481, 2.956, 2.542, 2.086, 1.502, 1.028, 0.7057,
 00952 0.5223, 0.4097, 0.3291, 0.2729, 0.2417, 0.2235, 0.2035, 0.2565, 0.2228,
 00953 0.274, 0.339, 0.387, 0.3918, 0.396, 0.3931, 0.3692},
 00954 {3.447, 3.508, 3.934, 4.533, 5.226, 5.96, 6.611, 7.003, 6.968,
 00955 6.331, 5.301, 4.324, 3.576, 3.009, 2.45, 1.808, 1.285, 0.894,
 00956 0.6612, 0.5244, 0.4273, 0.3588, 0.3341, 0.3096, 0.296, 0.4036, 0.403,
 00957 0.5114, 0.6763, 0.7735, 0.7969, 0.8461, 0.9229, 0.9621},
 00958 {3.722, 3.706, 4.103, 4.704, 5.429, 6.225, 6.997, 7.531, 7.688,
 00959 7.197, 6.151, 5.021, 4.119, 3.436, 2.786, 2.084, 1.513, 1.082,
 00960 0.808, 0.6428, 0.5308, 0.473, 0.5475, 0.6436, 0.4784, 0.5735, 0.5953,
 00961 0.7959, 1.106, 1.287, 1.339, 1.478, 1.689, 1.86},
 00962 {3.921, 3.841, 4.189, 4.783, 5.529, 6.4, 7.263, 7.978, 8.293,
 00963 7.875, 6.788, 5.549, 4.498, 3.668, 2.902, 2.117, 1.551, 1.158,

```

00964      0.9124, 0.7546, 0.6628, 0.6526, 0.7867, 0.9666, 0.4912, 0.664, 0.8688,
00965      1.172, 1.475, 1.537, 1.57, 1.791, 2.33, 2.77},
00966      {4.115, 3.995, 4.345, 4.915, 5.667, 6.583, 7.547, 8.4, 8.806,
00967      8.416, 7.274, 5.963, 4.777, 3.806, 2.936, 2.114, 1.555, 1.194,
00968      0.9788, 0.8251, 0.7444, 0.7323, 0.8067, 0.4073, 0.4371, 0.8223, 1.294,
00969      1.799, 1.845, 1.715, 1.83, 2.164, 2.718, 3.019},
00970      {4.267, 4.151, 4.545, 5.138, 5.928, 6.9, 7.906, 8.818, 9.276,
00971      8.888, 7.7, 6.275, 4.923, 3.83, 2.949, 2.126, 1.57, 1.211,
00972      0.9919, 0.8349, 0.7611, 0.7613, 0.8757, 0.4158, 0.6205, 0.9285, 1.474,
00973      1.966, 1.886, 1.546, 1.534, 1.58, 1.727, 1.781},
00974      {4.357, 4.249, 4.661, 5.37, 6.245, 7.245, 8.255, 9.142, 9.591,
00975      9.187, 7.974, 6.489, 5.106, 3.983, 3.055, 2.223, 1.646, 1.263,
00976      1.021, 0.8595, 0.7835, 0.7906, 0.9308, 0.4752, 0.6857, 1.015, 1.644,
00977      2.238, 2.142, 1.682, 1.628, 1.619, 1.797, 1.869},
00978      {4.506, 4.431, 4.893, 5.628, 6.535, 7.541, 8.516, 9.354, 9.751,
00979      9.305, 8.052, 6.578, 5.209, 4.113, 3.227, 2.379, 1.745, 1.32,
00980      1.053, 0.8816, 0.796, 0.7899, 0.9061, 1.191, 0.5672, 0.9669, 1.349,
00981      1.746, 1.888, 1.592, 1.62, 1.791, 2.222, 2.485},
00982      {4.576, 4.506, 4.983, 5.745, 6.664, 7.67, 8.657, 9.446, 9.792,
00983      9.306, 8.032, 6.569, 5.251, 4.219, 3.374, 2.515, 1.835, 1.359,
00984      1.057, 0.8733, 0.7654, 0.7338, 0.8346, 1.067, 0.6038, 1.054, 1.38,
00985      1.684, 1.756, 1.589, 1.679, 2.018, 2.626, 2.986},
00986      {4.675, 4.625, 5.078, 5.828, 6.714, 7.661, 8.588, 9.4, 9.761,
00987      9.26, 7.973, 6.455, 5.133, 4.127, 3.339, 2.562, 1.884, 1.37,
00988      1.036, 0.8316, 0.7014, 0.6314, 0.6329, 0.6673, 0.7434, 1.216, 1.142,
00989      1.346, 1.515, 1.614, 1.773, 2.186, 2.758, 3.13},
00990      {4.664, 4.649, 5.144, 5.923, 6.826, 7.78, 8.69, 9.369, 9.701,
00991      9.297, 8.02, 6.479, 5.147, 4.209, 3.474, 2.764, 2.093, 1.519,
00992      1.113, 0.8486, 0.683, 0.5849, 0.5324, 0.5209, 0.5327, 1.058, 0.9241,
00993      1.048, 1.287, 1.427, 1.546, 1.782, 2.111, 2.301},
00994      {4.771, 4.822, 5.408, 6.232, 7.155, 8.095, 8.92, 9.55, 9.74,
00995      9.14, 7.776, 6.287, 5.065, 4.198, 3.518, 2.824, 2.145, 1.537,
00996      1.096, 0.8121, 0.644, 0.5446, 0.4895, 0.4683, 0.463, 0.4501, 0.773,
00997      0.7994, 0.9657, 1.138, 1.222, 1.365, 1.564, 1.666},
00998      {5.238, 5.493, 6.287, 7.287, 8.326, 9.282, 10.02, 10.41, 10.22,
00999      9.194, 7.64, 6.172, 5.058, 4.272, 3.57, 2.766, 1.962, 1.313,
01000      0.8882, 0.6342, 0.4953, 0.4159, 0.3845, 0.3668, 0.3524, 0.3383, 0.6226,
01001      0.6524, 0.702, 0.662, 0.6453, 0.6304, 0.6111, 0.5462},
01002      {5.459, 5.786, 6.717, 8.018, 9.426, 10.72, 11.67, 11.98, 11.43,
01003      9.923, 8.082, 6.51, 5.425, 4.719, 3.958, 2.937, 1.953, 1.26,
01004      0.8432, 0.5868, 0.4389, 0.3576, 0.3281, 0.3182, 0.296, 0.2263, 0.5208,
01005      0.5264, 0.5272, 0.439, 0.4076, 0.3774, 0.3392, 0.2954}},
01006      {{1.93, 2.082, 2.236, 2.401, 2.486, 2.46, 2.242, 1.936, 1.632,
01007      1.309, 1.205, 0.996, 0.8843, 0.5832, 0.3788, 0.2472, 0.1935, 0.199,
01008      0.2177, 0.3668, 0.2468, 0.1727, 0.1235, 0.1211, 0.09577, 0.05738, 0.01593,
01009      0.01572, 0.01585, 0.01477, 0.01213, 0.01077, 0.008684, 0.005934},
01010      {2.406, 2.637, 3.006, 3.467, 3.941, 4.339, 4.464, 4.221, 3.692,
01011      2.961, 2.333, 1.856, 1.579, 1.321, 0.9877, 0.7954, 0.535, 0.3953,
01012      0.3269, 0.3153, 0.4016, 0.4948, 0.4946, 0.3969, 0.2986, 0.157, 0.08327,
01013      0.09294, 0.1047, 0.09143, 0.08129, 0.06982, 0.05108, 0.0324},
01014      {2.891, 3.082, 3.531, 4.114, 4.759, 5.387, 5.81, 5.856, 5.447,
01015      4.602, 3.716, 2.967, 2.422, 1.997, 1.543, 1.312, 1.043, 0.7202,
01016      0.5009, 0.3828, 0.3369, 0.3204, 0.3053, 0.2956, 0.2344, 0.3256, 0.2033,
01017      0.2183, 0.2574, 0.252, 0.2454, 0.2496, 0.2494, 0.2289},
01018      {3.257, 3.412, 3.896, 4.558, 5.307, 6.077, 6.732, 7.047, 6.849,
01019      6.037, 4.935, 3.973, 3.242, 2.755, 2.32, 1.807, 1.313, 0.9215,
01020      0.6619, 0.4999, 0.3902, 0.3134, 0.2686, 0.245, 0.2368, 0.2048, 0.1813,
01021      0.2732, 0.3298, 0.3568, 0.3526, 0.3493, 0.3549, 0.3469},
01022      {3.587, 3.682, 4.173, 4.839, 5.622, 6.472, 7.24, 7.722, 7.706,
01023      6.99, 5.826, 4.706, 3.861, 3.255, 2.675, 1.997, 1.417, 0.9866,
01024      0.7187, 0.561, 0.4546, 0.3832, 0.3527, 0.3395, 0.3279, 0.4213, 0.3649,
01025      0.4532, 0.5745, 0.6214, 0.6234, 0.6274, 0.6214, 0.5786},
01026      {3.86, 3.88, 4.313, 4.965, 5.741, 6.605, 7.439, 8.066, 8.231,
01027      7.669, 6.514, 5.29, 4.308, 3.595, 2.953, 2.22, 1.613, 1.142,
01028      0.8371, 0.6574, 0.5366, 0.4574, 0.4295, 0.4016, 0.3794, 0.5616, 0.5829,
01029      0.7168, 0.9521, 1.025, 1.053, 1.129, 1.25, 1.317},
01030      {4.081, 4.041, 4.427, 5.032, 5.788, 6.668, 7.53, 8.225, 8.513,
01031      8.083, 6.997, 5.729, 4.642, 3.826, 3.135, 2.369, 1.741, 1.266,
01032      0.9467, 0.7487, 0.6194, 0.5505, 0.6097, 0.7323, 0.5351, 0.6646, 0.7098,
01033      0.9842, 1.318, 1.49, 1.534, 1.718, 2.035, 2.285},
01034      {4.216, 4.126, 4.503, 5.084, 5.827, 6.712, 7.64, 8.42, 8.793,
01035      8.425, 7.34, 6.028, 4.838, 3.895, 3.074, 2.264, 1.681, 1.272,
01036      1.009, 0.8308, 0.7345, 0.7299, 0.8734, 1.142, 0.5401, 0.7387, 0.9717,
01037      1.328, 1.652, 1.667, 1.69, 1.954, 2.622, 3.19},
01038      {4.272, 4.151, 4.513, 5.089, 5.851, 6.773, 7.683, 8.504, 8.944,
01039      8.636, 7.572, 6.215, 4.931, 3.883, 2.974, 2.147, 1.591, 1.231,
01040      1.009, 0.859, 0.7919, 0.8024, 0.896, 0.4226, 0.5516, 0.8571, 1.354,
01041      1.883, 1.918, 1.778, 1.904, 2.264, 2.865, 3.19},
01042      {4.268, 4.149, 4.512, 5.115, 5.895, 6.826, 7.773, 8.616, 9.068,
01043      8.77, 7.691, 6.307, 4.976, 3.859, 2.932, 2.123, 1.578, 1.222,
01044      0.9995, 0.8488, 0.7895, 0.8146, 0.9335, 0.4125, 0.6071, 0.932, 1.493,
01045      1.986, 1.896, 1.55, 1.538, 1.583, 1.731, 1.792},
01046      {4.24, 4.152, 4.565, 5.21, 6.012, 6.947, 7.876, 8.687, 9.116,
01047      8.769, 7.647, 6.266, 4.953, 3.871, 2.972, 2.151, 1.594, 1.22,
01048      0.9831, 0.8291, 0.7694, 0.7899, 0.9141, 0.4431, 0.6319, 0.9495, 1.544,
01049      2.116, 2.045, 1.603, 1.547, 1.538, 1.698, 1.778},
01050      {4.22, 4.152, 4.587, 5.265, 6.061, 6.961, 7.879, 8.672, 9.045,

```

01051 8.614, 7.444, 6.083, 4.845, 3.844, 2.996, 2.197, 1.608, 1.21,
 01052 0.9591, 0.8021, 0.7294, 0.7305, 0.8376, 1.102, 0.495, 0.808, 1.134,
 01053 1.49, 1.689, 1.458, 1.475, 1.619, 1.978, 2.214},
 01054 {4.139, 4.115, 4.593, 5.262, 6.061, 6.95, 7.815, 8.557, 8.84,
 01055 8.311, 7.1, 5.777, 4.638, 3.742, 2.972, 2.201, 1.594, 1.167,
 01056 0.9052, 0.7436, 0.6504, 0.62, 0.7047, 0.8275, 0.483, 0.7578, 0.9831,
 01057 1.246, 1.432, 1.377, 1.433, 1.677, 2.116, 2.395},
 01058 {4.027, 4.03, 4.521, 5.195, 5.986, 6.839, 7.642, 8.241, 8.387,
 01059 7.78, 6.581, 5.331, 4.311, 3.538, 2.878, 2.173, 1.57, 1.118,
 01060 0.8368, 0.6691, 0.5655, 0.5116, 0.5213, 0.5546, 0.5962, 0.7812, 0.7265,
 01061 0.8823, 1.107, 1.292, 1.385, 1.633, 2.016, 2.214},
 01062 {3.787, 3.852, 4.369, 5.07, 5.845, 6.625, 7.28, 7.735, 7.755,
 01063 7.082, 5.9, 4.755, 3.857, 3.202, 2.64, 2.031, 1.478, 1.036,
 01064 0.7546, 0.5863, 0.4836, 0.422, 0.3872, 0.3827, 0.4058, 0.6138, 0.5067,
 01065 0.6048, 0.7857, 0.9735, 1.052, 1.172, 1.322, 1.358},
 01066 {3.556, 3.717, 4.24, 4.935, 5.651, 6.309, 6.815, 7.1, 6.959,
 01067 6.199, 5.099, 4.12, 3.39, 2.875, 2.37, 1.798, 1.291, 0.8979,
 01068 0.6433, 0.4909, 0.3991, 0.3437, 0.3063, 0.2936, 0.2983, 0.283, 0.3393,
 01069 0.3825, 0.4825, 0.6185, 0.6697, 0.7099, 0.7581, 0.7492},
 01070 {3.344, 3.611, 4.169, 4.812, 5.45, 5.983, 6.257, 6.263, 5.844,
 01071 4.989, 4.049, 3.337, 2.825, 2.403, 1.893, 1.346, 0.9248, 0.6385,
 01072 0.4618, 0.3493, 0.281, 0.2397, 0.2114, 0.1989, 0.1936, 0.1739, 0.1941,
 01073 0.2137, 0.2382, 0.2613, 0.2565, 0.2526, 0.2392, 0.2172},
 01074 {3.617, 4.108, 4.785, 5.409, 5.873, 6.058, 5.747, 5.269, 4.577,
 01075 3.776, 3.142, 2.721, 2.304, 1.764, 1.195, 0.7421, 0.4648, 0.3052,
 01076 0.2132, 0.1552, 0.1201, 0.1028, 0.09496, 0.09272, 0.092, 0.06578, 0.09663,
 01077 0.1039, 0.1161, 0.1127, 0.1081, 0.1019, 0.09228, 0.08144}},
 01078 {{4.776, 5.263, 6.105, 7.209, 8.284, 9.138, 9.553, 9.22, 8.161,
 01079 6.444, 5.129, 4.002, 3.207, 2.842, 2.466, 1.792, 1.171, 0.7381,
 01080 0.5022, 0.4253, 0.4401, 0.5454, 0.7785, 0.6418, 0.4899, 0.3887, 0.3859,
 01081 0.2463, 0.248, 0.1831, 0.1414, 0.1233, 0.104, 0.07015},
 01082 {4.315, 4.602, 5.253, 6.073, 6.933, 7.732, 8.306, 8.428, 7.938,
 01083 6.819, 5.489, 4.358, 3.446, 2.895, 2.476, 1.885, 1.473, 1.07,
 01084 0.7373, 0.5219, 0.4618, 0.454, 0.4569, 0.4263, 0.397, 0.4683, 0.4728,
 01085 0.4348, 0.4638, 0.3711, 0.3221, 0.2786, 0.2347, 0.153},
 01086 {4.034, 4.197, 4.772, 5.576, 6.46, 7.358, 8.131, 8.568, 8.457,
 01087 7.592, 6.31, 5.095, 4.065, 3.363, 2.711, 2.026, 1.686, 1.33,
 01088 1.006, 0.7144, 0.5379, 0.4588, 0.4427, 0.451, 0.4738, 0.5938, 0.595,
 01089 0.5555, 0.6354, 0.6081, 0.5877, 0.5953, 0.6217, 0.6394},
 01090 {4.052, 4.154, 4.699, 5.472, 6.377, 7.353, 8.242, 8.879, 9.016,
 01091 8.382, 7.119, 5.777, 4.673, 3.861, 3.197, 2.408, 1.717, 1.19,
 01092 0.8643, 0.6598, 0.5193, 0.4314, 0.3937, 0.3863, 0.4011, 0.3591, 0.4105,
 01093 0.4673, 0.5539, 0.5717, 0.5531, 0.5518, 0.5563, 0.5379},
 01094 {4.153, 4.213, 4.735, 5.466, 6.334, 7.3, 8.25, 8.998, 9.286,
 01095 8.779, 7.558, 6.165, 5, 4.154, 3.419, 2.575, 1.849, 1.298,
 01096 0.9519, 0.7439, 0.6103, 0.537, 0.545, 0.5816, 0.5683, 0.7049, 0.6064,
 01097 0.7504, 0.9181, 0.9021, 0.8857, 0.8707, 0.8466, 0.7728},
 01098 {4.248, 4.254, 4.738, 5.442, 6.269, 7.19, 8.14, 8.939, 9.34,
 01099 8.978, 7.82, 6.411, 5.184, 4.273, 3.525, 2.664, 1.942, 1.394,
 01100 1.033, 0.8143, 0.6732, 0.5954, 0.6154, 0.6352, 0.7198, 0.8339, 0.8119,
 01101 1.046, 1.311, 1.244, 1.264, 1.359, 1.508, 1.629},
 01102 {4.354, 4.315, 4.761, 5.42, 6.219, 7.126, 8.036, 8.844, 9.254,
 01103 8.926, 7.833, 6.444, 5.174, 4.22, 3.457, 2.639, 1.948, 1.432,
 01104 1.087, 0.8685, 0.7349, 0.6908, 0.8, 0.9447, 0.6798, 0.8977, 1.108,
 01105 1.425, 1.666, 1.47, 1.449, 1.545, 1.768, 1.943},
 01106 {4.362, 4.274, 4.672, 5.312, 6.096, 7.012, 7.964, 8.785, 9.19,
 01107 8.837, 7.774, 6.356, 5.026, 3.988, 3.137, 2.353, 1.747, 1.332,
 01108 1.065, 0.8964, 0.8178, 0.823, 0.9551, 1.158, 0.5854, 0.9884, 1.483,
 01109 2.035, 2.091, 1.603, 1.521, 1.55, 1.723, 1.854},
 01110 {4.3, 4.174, 4.537, 5.167, 5.965, 6.89, 7.821, 8.663, 9.083,
 01111 8.735, 7.634, 6.25, 4.936, 3.878, 2.975, 2.165, 1.611, 1.244,
 01112 1.01, 0.864, 0.8216, 0.8541, 0.947, 0.4242, 0.4811, 0.9771, 1.569,
 01113 2.131, 2.042, 1.616, 1.566, 1.597, 1.77, 1.859},
 01114 {4.191, 4.082, 4.474, 5.086, 5.853, 6.754, 7.678, 8.491, 8.887,
 01115 8.503, 7.408, 6.052, 4.775, 3.739, 2.846, 2.066, 1.531, 1.182,
 01116 0.9574, 0.8217, 0.7847, 0.8129, 0.9004, 0.3845, 0.5912, 0.9488, 1.54,
 01117 2.073, 1.987, 1.591, 1.544, 1.55, 1.68, 1.746},
 01118 {4.067, 4, 4.384, 4.981, 5.736, 6.618, 7.486, 8.26, 8.602,
 01119 8.169, 7.063, 5.748, 4.553, 3.579, 2.743, 1.997, 1.466, 1.117,
 01120 0.8947, 0.756, 0.7085, 0.7246, 0.8226, 0.3804, 0.4137, 0.8378, 1.338,
 01121 1.831, 1.823, 1.452, 1.396, 1.406, 1.546, 1.607},
 01122 {3.852, 3.824, 4.228, 4.825, 5.559, 6.404, 7.233, 7.922, 8.179,
 01123 7.678, 6.56, 5.348, 4.269, 3.383, 2.62, 1.89, 1.369, 1.029,
 01124 0.8128, 0.6781, 0.613, 0.6018, 0.6705, 0.6956, 0.4126, 0.6338, 0.8946,
 01125 1.186, 1.405, 1.278, 1.277, 1.398, 1.661, 1.83},
 01126 {3.577, 3.6, 4.019, 4.632, 5.336, 6.114, 6.899, 7.485, 7.599,
 01127 6.995, 5.898, 4.813, 3.907, 3.169, 2.502, 1.82, 1.298, 0.9403,
 01128 0.7213, 0.5834, 0.4987, 0.4571, 0.4767, 0.4929, 0.357, 0.5342, 0.6218,
 01129 0.8321, 1.072, 1.133, 1.182, 1.326, 1.548, 1.687},
 01130 {3.251, 3.338, 3.79, 4.396, 5.094, 5.833, 6.465, 6.856, 6.763,
 01131 6.045, 5.018, 4.111, 3.394, 2.814, 2.265, 1.671, 1.175, 0.8195,
 01132 0.6074, 0.4777, 0.3953, 0.3477, 0.3295, 0.3281, 0.3202, 0.3996, 0.4085,
 01133 0.5264, 0.6945, 0.8446, 0.9062, 1.002, 1.145, 1.182},
 01134 {2.844, 3.014, 3.474, 4.053, 4.689, 5.317, 5.776, 5.907, 5.606,
 01135 4.864, 3.98, 3.263, 2.726, 2.292, 1.824, 1.308, 0.8985, 0.62,
 01136 0.4532, 0.3508, 0.2879, 0.2496, 0.2268, 0.217, 0.2089, 0.1723, 0.2384,
 01137 0.3033, 0.3816, 0.457, 0.4738, 0.4945, 0.522, 0.511},

```

01138 {2.32, 2.558, 2.981, 3.486, 3.981, 4.383, 4.505, 4.35, 3.92,
01139 3.298, 2.688, 2.238, 1.901, 1.574, 1.197, 0.8142, 0.5528, 0.3893,
01140 0.2874, 0.2207, 0.179, 0.153, 0.1353, 0.126, 0.118, 0.0996, 0.1038,
01141 0.1254, 0.1556, 0.1673, 0.1637, 0.1606, 0.1599, 0.1512},
01142 {1.604, 1.812, 2.085, 2.33, 2.517, 2.565, 2.341, 2.07, 1.768,
01143 1.479, 1.254, 1.072, 0.8657, 0.6304, 0.4269, 0.2773, 0.1887, 0.1365,
01144 0.104, 0.08106, 0.06679, 0.05742, 0.0504, 0.04538, 0.04064, 0.03211,
01145 0.02611,
01146 0.02727, 0.03066, 0.03125, 0.02913, 0.02667, 0.02485, 0.02217},
01147 {0.4429, 0.4652, 0.4579, 0.4357, 0.388, 0.3401, 0.2901, 0.2551, 0.2192,
01148 0.1788, 0.1348, 0.0914, 0.05502, 0.03148, 0.01858, 0.01216, 0.009078,
01149 0.007534,
01150 0.006492, 0.006257, 0.006301, 0.006115, 0.005378, 0.005084, 0.005022,
01151 0.002766, 0.00246,
01152 0.001431, 0.001208, 0.0014, 0.001257, 0.001091, 0.0009076, 0.0005632}},
01153 {{6.159, 6.267, 7.137, 8.441, 9.868, 11.32, 12.68, 13.44, 13.31,
01154 11.97, 9.899, 7.982, 6.529, 5.456, 4.582, 3.411, 2.447, 1.735,
01155 1.201, 0.8902, 0.7385, 0.82, 0.8247, 0.6792, 0.5978, 0.7594, 1.311,
01156 0.8001, 0.7222, 0.5196, 0.4, 0.3513, 0.3049, 0.2033},
01157 {6.025, 6.218, 7.108, 8.305, 9.599, 10.86, 11.89, 12.5, 12.4,
01158 11.24, 9.372, 7.562, 6.173, 5.198, 4.359, 3.187, 2.293, 1.674,
01159 1.209, 0.8758, 0.6848, 0.7086, 0.7544, 0.6892, 0.6462, 0.8934, 1.25,
01160 0.9268, 0.9078, 0.7087, 0.6172, 0.5399, 0.4671, 0.304},
01161 {5.423, 5.508, 6.112, 6.997, 8.017, 9.107, 10.07, 10.75, 10.95,
01162 10.26, 8.775, 7.139, 5.746, 4.723, 3.902, 2.898, 2.14, 1.579,
01163 1.155, 0.86, 0.6747, 0.5805, 0.5689, 0.5958, 0.6918, 0.8863, 1.028,
01164 0.8369, 0.8899, 0.8223, 0.7867, 0.7493, 0.7297, 0.6911},
01165 {5.03, 5.007, 5.533, 6.337, 7.269, 8.306, 9.353, 10.16, 10.53,
01166 10.09, 8.802, 7.176, 5.772, 4.785, 3.999, 3.116, 2.292, 1.613,
01167 1.161, 0.8825, 0.7044, 0.602, 0.5675, 0.5799, 0.633, 0.5335, 0.75,
01168 0.782, 0.8887, 0.8566, 0.8053, 0.779, 0.7512, 0.6894},
01169 {4.854, 4.795, 5.239, 5.991, 6.89, 7.916, 8.949, 9.794, 10.25,
01170 9.932, 8.787, 7.214, 5.806, 4.824, 4.046, 3.151, 2.318, 1.656,
01171 1.216, 0.9337, 0.7505, 0.6429, 0.6093, 0.6241, 0.6313, 0.828, 0.8414,
01172 1.007, 1.191, 1.083, 1.037, 1.002, 0.963, 0.8665},
01173 {4.689, 4.616, 5.08, 5.801, 6.661, 7.626, 8.619, 9.471, 9.972,
01174 9.711, 8.582, 7.039, 5.65, 4.664, 3.923, 3.04, 2.236, 1.63,
01175 1.223, 0.9546, 0.7831, 0.6819, 0.6534, 0.6599, 0.5823, 0.9021, 1.016,
01176 1.267, 1.495, 1.395, 1.398, 1.508, 1.711, 1.859},
01177 {4.575, 4.466, 4.925, 5.634, 6.479, 7.412, 8.343, 9.218, 9.74,
01178 9.468, 8.319, 6.801, 5.4, 4.37, 3.571, 2.713, 2.014, 1.498,
01179 1.158, 0.9364, 0.7954, 0.7273, 0.7681, 0.8566, 0.6687, 0.9463, 1.219,
01180 1.575, 1.788, 1.546, 1.525, 1.635, 1.887, 2.09},
01181 {4.408, 4.299, 4.735, 5.433, 6.275, 7.22, 8.175, 9.032, 9.47,
01182 9.097, 7.942, 6.45, 5.08, 4.019, 3.16, 2.338, 1.735, 1.324,
01183 1.063, 0.9095, 0.8421, 0.8249, 0.8269, 0.6041, 0.6079, 1.07, 1.601,
01184 2.169, 2.204, 1.68, 1.593, 1.624, 1.805, 1.939},
01185 {4.26, 4.136, 4.516, 5.172, 5.992, 6.946, 7.925, 8.739, 9.102,
01186 8.667, 7.534, 6.13, 4.865, 3.837, 2.924, 2.126, 1.572, 1.207,
01187 0.9693, 0.8391, 0.8016, 0.8213, 0.8584, 0.4114, 0.632, 0.9986, 1.59,
01188 2.132, 2.045, 1.617, 1.567, 1.595, 1.754, 1.846},
01189 {4.068, 3.944, 4.321, 4.953, 5.759, 6.696, 7.598, 8.349, 8.636,
01190 8.134, 6.966, 5.681, 4.531, 3.592, 2.726, 1.981, 1.454, 1.112,
01191 0.8863, 0.7642, 0.7336, 0.7503, 0.7883, 0.3649, 0.5742, 0.9245, 1.482,
01192 1.979, 1.91, 1.537, 1.492, 1.497, 1.609, 1.655},
01193 {3.784, 3.702, 4.08, 4.686, 5.436, 6.274, 7.107, 7.772, 7.978,
01194 7.457, 6.38, 5.207, 4.171, 3.317, 2.572, 1.856, 1.347, 1.012,
01195 0.8031, 0.6877, 0.6459, 0.6441, 0.6934, 0.3392, 0.5146, 0.7711, 1.206,
01196 1.627, 1.668, 1.358, 1.31, 1.315, 1.411, 1.432},
01197 {3.39, 3.409, 3.832, 4.448, 5.159, 5.932, 6.66, 7.149, 7.206,
01198 6.618, 5.602, 4.608, 3.743, 3.01, 2.347, 1.689, 1.206, 0.8923,
01199 0.6966, 0.5763, 0.5136, 0.4878, 0.5216, 0.5783, 0.3499, 0.515, 0.7012,
01200 0.9131, 1.167, 1.133, 1.139, 1.212, 1.359, 1.445},
01201 {3.031, 3.122, 3.551, 4.115, 4.781, 5.496, 6.101, 6.433, 6.32,
01202 5.654, 4.707, 3.886, 3.211, 2.629, 2.053, 1.473, 1.024, 0.7318,
01203 0.5579, 0.445, 0.3748, 0.3356, 0.3272, 0.3261, 0.3502, 0.4067, 0.4482,
01204 0.5625, 0.7534, 0.8328, 0.8615, 0.9261, 1.038, 1.075},
01205 {2.556, 2.697, 3.11, 3.64, 4.251, 4.887, 5.363, 5.492, 5.176,
01206 4.453, 3.662, 3.064, 2.599, 2.164, 1.677, 1.161, 0.7816, 0.5445,
01207 0.4076, 0.3171, 0.258, 0.2227, 0.2043, 0.1946, 0.1903, 0.2423, 0.2411,
01208 0.2984, 0.3661, 0.4305, 0.4483, 0.4735, 0.5096, 0.5082},
01209 {1.982, 2.163, 2.522, 2.962, 3.444, 3.894, 4.12, 3.996, 3.538,
01210 2.915, 2.39, 2.044, 1.761, 1.418, 1.026, 0.6684, 0.4452, 0.3147,
01211 0.2354, 0.1814, 0.1474, 0.1272, 0.1136, 0.1042, 0.09334, 0.07244, 0.09453,
01212 0.1067, 0.1323, 0.1309, 0.1255, 0.1235, 0.1251, 0.1207},
01213 {1.313, 1.48, 1.706, 1.932, 2.113, 2.193, 2.081, 1.804, 1.487,
01214 1.196, 0.9808, 0.8365, 0.6791, 0.4931, 0.3304, 0.2112, 0.1439, 0.1054,
01215 0.08052, 0.06314, 0.05248, 0.04667, 0.0419, 0.03731, 0.03192, 0.02135,
01216 0.01682,
01217 0.0156, 0.01767, 0.01723, 0.0161, 0.01526, 0.0148, 0.01411},
01218 {0.242, 0.2311, 0.2162, 0.1962, 0.1752, 0.1604, 0.1387, 0.1112, 0.08183,
01219 0.05815, 0.04045, 0.02676, 0.01677, 0.01075, 0.007653, 0.005984, 0.00512,
01220 0.004795,
01221 0.004786, 0.004999, 0.004952, 0.004352, 0.003443, 0.002664, 0.002223,
01222 0.001163, 0.001542,
01223 0.0002821, 0.0001951, 0.000206, 0.0001656, 0.0001206, 8.303e-05,
01224 5.901e-05},

```

```

01225 {0.0001232, 0.0001559, 0.0001539, 0.0001693, 0.0002134, 0.0002031,
01226 0.0001037, 1.126e-05, 5.382e-06,
01227 1.867e-06, 5.983e-07, 2.464e-07, 1.576e-07, 1.322e-07, 1.312e-07,
01228 1.319e-07, 3.921e-07, 3.583e-06,
01229 3.815e-05, 6.754e-05, 0.0001004, 0.0002135, 0.0004217, 0.0007681,
01230 0.001524, 0.0004274, 0.000876,
01231 2.698e-05, 1.328e-12, 1.445e-13, 9.798e-14, 8.583e-14, 9.786e-14,
01232 1.774e-13}},
01233 {{6.595, 6.532, 7.313, 8.453, 9.864, 11.47, 13.06, 14.28, 14.67,
01234 13.68, 11.56, 9.275, 7.452, 6.201, 5.275, 4.16, 2.898, 2.003,
01235 1.4, 1.04, 0.7754, 0.7071, 0.7598, 0.799, 0.825, 0.9217, 1.851,
01236 1.254, 1.138, 0.8159, 0.6311, 0.5427, 0.4614, 0.3814},
01237 {6.516, 6.556, 7.327, 8.526, 9.924, 11.42, 12.85, 13.82, 14.03,
01238 13.05, 11.03, 8.863, 7.108, 5.878, 4.956, 3.797, 2.704, 1.92,
01239 1.344, 0.9685, 0.7276, 0.6364, 0.6746, 0.7239, 0.786, 0.9333, 1.793,
01240 1.344, 1.234, 0.8885, 0.7949, 0.6932, 0.5878, 0.4871},
01241 {6.179, 6.202, 6.853, 7.807, 8.924, 10.13, 11.21, 12.01, 12.29,
01242 11.63, 10.05, 8.152, 6.536, 5.386, 4.503, 3.473, 2.521, 1.809,
01243 1.273, 0.9058, 0.6837, 0.5774, 0.5746, 0.6269, 0.7726, 0.9434, 1.275,
01244 1.102, 1.148, 0.9922, 0.9195, 0.8713, 0.8162, 0.7358},
01245 {5.401, 5.302, 5.812, 6.634, 7.64, 8.785, 9.902, 10.82, 11.3,
01246 10.96, 9.696, 7.981, 6.412, 5.281, 4.41, 3.469, 2.606, 1.892,
01247 1.37, 1.034, 0.8087, 0.6766, 0.6565, 0.6981, 0.7901, 0.6904, 1.01,
01248 1.062, 1.192, 1.063, 1.016, 0.9639, 0.8911, 0.7914},
01249 {5.101, 4.973, 5.426, 6.18, 7.138, 8.24, 9.32, 10.29, 10.9,
01250 10.75, 9.665, 8.035, 6.469, 5.319, 4.452, 3.502, 2.649, 1.941,
01251 1.431, 1.09, 0.869, 0.7456, 0.7339, 0.7833, 0.8079, 1.059, 1.104,
01252 1.303, 1.515, 1.253, 1.185, 1.131, 1.076, 0.9437},
01253 {4.936, 4.795, 5.272, 5.985, 6.878, 7.91, 8.989, 9.922, 10.53,
01254 10.37, 9.278, 7.698, 6.176, 5.044, 4.178, 3.263, 2.472, 1.849,
01255 1.402, 1.087, 0.8859, 0.7846, 0.8226, 0.8854, 0.9635, 1.037, 1.251,
01256 1.527, 1.706, 1.5, 1.503, 1.644, 1.914, 2.113},
01257 {4.796, 4.617, 5.024, 5.703, 6.591, 7.617, 8.632, 9.544, 10.07,
01258 9.749, 8.552, 6.983, 5.55, 4.462, 3.573, 2.707, 2.021, 1.537,
01259 1.216, 1.017, 0.9039, 0.8702, 0.9836, 1.21, 0.6125, 1.009, 1.311,
01260 1.688, 1.862, 1.575, 1.568, 1.696, 2.001, 2.214},
01261 {4.522, 4.356, 4.742, 5.465, 6.36, 7.357, 8.359, 9.269, 9.706,
01262 9.237, 7.95, 6.476, 5.137, 4.086, 3.214, 2.373, 1.75, 1.33,
01263 1.071, 0.9379, 0.8929, 0.9071, 0.9736, 1.305, 0.5218, 1.054, 1.605,
01264 2.105, 1.976, 1.563, 1.521, 1.56, 1.765, 1.875},
01265 {4.201, 4.084, 4.453, 5.134, 5.998, 7.007, 8.042, 8.894, 9.218,
01266 8.665, 7.393, 5.966, 4.728, 3.77, 2.956, 2.16, 1.585, 1.199,
01267 0.9637, 0.8579, 0.8414, 0.8686, 0.8189, 1.154, 0.4693, 0.9934, 1.568,
01268 2.075, 1.962, 1.563, 1.524, 1.545, 1.704, 1.786},
01269 {3.87, 3.761, 4.135, 4.74, 5.547, 6.523, 7.533, 8.287, 8.542,
01270 7.978, 6.743, 5.463, 4.36, 3.491, 2.739, 1.993, 1.453, 1.095,
01271 0.8767, 0.7822, 0.7664, 0.777, 0.8145, 1.109, 0.4094, 0.8854, 1.413,
01272 1.91, 1.872, 1.47, 1.421, 1.428, 1.538, 1.583},
01273 {3.565, 3.517, 3.908, 4.525, 5.299, 6.159, 6.982, 7.581, 7.734,
01274 7.15, 6.028, 4.918, 3.993, 3.242, 2.541, 1.833, 1.321, 0.9862,
01275 0.7851, 0.6877, 0.6504, 0.6409, 0.6657, 0.7916, 0.3852, 0.627, 0.8774,
01276 1.306, 1.713, 1.397, 1.317, 1.308, 1.379, 1.377},
01277 {3.27, 3.307, 3.718, 4.324, 5.008, 5.72, 6.391, 6.82, 6.844,
01278 6.25, 5.256, 4.321, 3.562, 2.929, 2.309, 1.67, 1.183, 0.8581,
01279 0.6613, 0.5437, 0.4817, 0.4549, 0.4828, 0.4971, 0.343, 0.4517, 0.5928,
01280 0.7482, 1.114, 1.156, 1.127, 1.142, 1.266, 1.325},
01281 {2.881, 2.972, 3.365, 3.885, 4.479, 5.095, 5.612, 5.869, 5.739,
01282 5.109, 4.233, 3.497, 2.928, 2.45, 1.923, 1.37, 0.937, 0.6588,
01283 0.4974, 0.3913, 0.3216, 0.2799, 0.263, 0.2476, 0.2702, 0.3664, 0.3897,
01284 0.4754, 0.6181, 0.6968, 0.7144, 0.7507, 0.8199, 0.8256},
01285 {2.352, 2.522, 2.914, 3.377, 3.888, 4.391, 4.73, 4.773, 4.456,
01286 3.814, 3.103, 2.576, 2.19, 1.824, 1.372, 0.9129, 0.606, 0.4281,
01287 0.3241, 0.25, 0.1992, 0.1685, 0.1489, 0.1316, 0.116, 0.1598, 0.1448,
01288 0.1805, 0.2224, 0.2379, 0.2369, 0.2454, 0.2679, 0.2718},
01289 {1.666, 1.833, 2.135, 2.486, 2.847, 3.14, 3.202, 3.006, 2.612,
01290 2.127, 1.726, 1.486, 1.277, 0.9733, 0.6654, 0.4233, 0.2852, 0.2051,
01291 0.1537, 0.1174, 0.09422, 0.08017, 0.06975, 0.06009, 0.04775, 0.03319,
01292 0.03371,
01293 0.03896, 0.04544, 0.04203, 0.03927, 0.03814, 0.03917, 0.04012},
01294 {0.8975, 0.9719, 1.03, 1.066, 1.034, 0.9374, 0.7957, 0.662, 0.5656,
01295 0.4856, 0.4141, 0.3239, 0.2283, 0.1478, 0.09439, 0.06056, 0.04188,
01296 0.03179,
01297 0.02625, 0.02293, 0.02134, 0.01999, 0.01796, 0.01508, 0.01136, 0.006243,
01298 0.005399,
01299 0.002554, 0.002671, 0.002877, 0.002693, 0.002456, 0.002169, 0.001592},
01300 {0.005568, 0.003081, 0.001936, 0.001388, 0.001138, 0.0009141, 0.0003913,
01301 7.042e-05, 1.305e-05,
01302 9.014e-06, 5.819e-06, 3.047e-06, 1.303e-06, 5.602e-07, 2.183e-07,
01303 1.757e-07, 3.825e-07, 2.566e-06,
01304 1.334e-05, 1.436e-05, 1.976e-05, 7.261e-05, 0.0002657, 0.0005962,
01305 0.001653, 0.0002773, 0.0008521,
01306 1.309e-06, 3.72e-14, 2.315e-16, 2.404e-15, 7.283e-17, 5.816e-17,
01307 1.165e-16},
01308 {5.606e-05, 7.174e-05, 7.065e-05, 8.779e-05, 0.0001175, 0.0001418,
01309 6.181e-05, 7.462e-06, 8.135e-06,
01310 6.922e-06, 3.21e-06, 1.063e-06, 3.185e-07, 7.307e-08, 1.298e-08,
01311 1.751e-08, 6.792e-08, 5.277e-07,

```

```

01312     7.612e-06, 1.832e-05, 4.78e-05, 0.0001019, 0.0001703, 0.0003801, 0.001213,
01313     0.0002105, 0.0006011,
01314     2.875e-06, 7.798e-13, 1.214e-13, 8.329e-14, 7.553e-14, 1.014e-13,
01315     1.901e-13}}
01316 };
01317
01318 #ifdef _OPENACC
01319 #pragma acc declare copyin(clim_oh_var)
01320 #endif
01321
01322 double clim_oh(
01323     double t,
01324     double lat,
01325     double p) {
01326
01327     /* Get seconds since begin of year... */
01328     double sec = FMOD(t, 365.25 * 86400.);
01329     while (sec < 0)
01330         sec += 365.25 * 86400.;
01331
01332     /* Check pressure... */
01333     if (p < clim_oh_ps[0])
01334         p = clim_oh_ps[0];
01335     else if (p > clim_oh_ps[33])
01336         p = clim_oh_ps[33];
01337
01338     /* Get indices... */
01339     int isec = locate_irr(clim_oh_secs, 12, sec);
01340     int ilat = locate_reg(clim_oh_lats, 18, lat);
01341     int ip = locate_irr(clim_oh_ps, 34, p);
01342
01343     /* Interpolate OH climatology (Pommrich et al., 2014)... */
01344     double aux00 = LIN(clim_oh_ps[ip],
01345         clim_oh_var[isec][ilat][ip],
01346         clim_oh_ps[ip + 1],
01347         clim_oh_var[isec][ilat][ip + 1], p);
01348     double aux01 = LIN(clim_oh_ps[ip],
01349         clim_oh_var[isec][ilat + 1][ip],
01350         clim_oh_ps[ip + 1],
01351         clim_oh_var[isec][ilat + 1][ip + 1], p);
01352     double aux10 = LIN(clim_oh_ps[ip],
01353         clim_oh_var[isec + 1][ilat][ip],
01354         clim_oh_ps[ip + 1],
01355         clim_oh_var[isec + 1][ilat][ip + 1], p);
01356     double aux11 = LIN(clim_oh_ps[ip],
01357         clim_oh_var[isec + 1][ilat + 1][ip],
01358         clim_oh_ps[ip + 1],
01359         clim_oh_var[isec + 1][ilat + 1][ip + 1], p);
01360     aux00 = LIN(clim_oh_lats[ilat], aux00, clim_oh_lats[ilat + 1], aux01, lat);
01361     aux11 = LIN(clim_oh_lats[ilat], aux10, clim_oh_lats[ilat + 1], aux11, lat);
01362     return 1e6 * LIN(clim_oh_secs[isec], aux00,
01363         clim_oh_secs[isec + 1], aux11, sec);
01364 }
01365
01366 /*****
01367
01368 static double clim_tropo_secs[12] = {
01369     1209600.00, 3888000.00, 6393600.00,
01370     9072000.00, 11664000.00, 14342400.00,
01371     16934400.00, 19612800.00, 22291200.00,
01372     24883200.00, 27561600.00, 30153600.00
01373 };
01374
01375 #ifdef _OPENACC
01376 #pragma acc declare copyin(clim_tropo_secs)
01377 #endif
01378
01379 static double clim_tropo_lats[73]
01380 = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
01381     -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
01382     -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
01383     -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
01384     15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
01385     45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
01386     75, 77.5, 80, 82.5, 85, 87.5, 90
01387 };
01388
01389 #ifdef _OPENACC
01390 #pragma acc declare copyin(clim_tropo_lats)
01391 #endif
01392
01393 static double clim_tropo_tps[12][73]
01394 = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
01395     297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
01396     175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
01397     99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
01398     98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,

```

01399 152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
 01400 277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
 01401 275.3, 275.6, 275.4, 274.1, 273.5},
 01402 {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
 01403 300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
 01404 150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
 01405 98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
 01406 98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
 01407 220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
 01408 284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
 01409 287.5, 286.2, 285.8},
 01410 {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
 01411 297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
 01412 161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
 01413 100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
 01414 99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
 01415 186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
 01416 279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
 01417 304.3, 304.9, 306, 306.6, 306.2, 306},
 01418 {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
 01419 290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
 01420 195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
 01421 102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
 01422 99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
 01423 148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
 01424 263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
 01425 315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
 01426 {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
 01427 260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
 01428 205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
 01429 101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
 01430 102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
 01431 165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
 01432 273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
 01433 325.3, 325.8, 325.8},
 01434 {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
 01435 222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
 01436 228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
 01437 105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
 01438 106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
 01439 127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
 01440 251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
 01441 308.5, 312.2, 313.1, 313.3},
 01442 {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
 01443 187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
 01444 235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
 01445 110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
 01446 111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
 01447 117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
 01448 224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
 01449 275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
 01450 {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
 01451 185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
 01452 233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
 01453 110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
 01454 112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
 01455 120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
 01456 230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
 01457 278.2, 282.6, 287.4, 290.9, 292.5, 293},
 01458 {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
 01459 183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
 01460 243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
 01461 114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
 01462 110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
 01463 114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
 01464 203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
 01465 276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
 01466 {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
 01467 215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
 01468 237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
 01469 111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
 01470 106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
 01471 112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
 01472 206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
 01473 279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
 01474 305.1},
 01475 {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
 01476 253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
 01477 223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
 01478 108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
 01479 102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
 01480 109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
 01481 241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
 01482 286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
 01483 {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
 01484 284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
 01485 175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,


```

01486 100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
01487 100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
01488 186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
01489 280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
01490 281.7, 281.1, 281.2}
01491 };
01492
01493 #ifdef _OPENACC
01494 #pragma acc declare copyin(clim_tropo_tps)
01495 #endif
01496
01497 double clim_tropo(
01498     double t,
01499     double lat) {
01500
01501     /* Get seconds since begin of year... */
01502     double sec = FMOD(t, 365.25 * 86400.);
01503     while (sec < 0)
01504         sec += 365.25 * 86400.;
01505
01506     /* Get indices... */
01507     int isec = locate_irr(clim_tropo_secs, 12, sec);
01508     int ilat = locate_reg(clim_tropo_lats, 73, lat);
01509
01510     /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
01511     double p0 = LIN(clim_tropo_lats[ilat],
01512                     clim_tropo_tps[isec][ilat],
01513                     clim_tropo_lats[ilat + 1],
01514                     clim_tropo_tps[isec][ilat + 1], lat);
01515     double p1 = LIN(clim_tropo_lats[ilat],
01516                     clim_tropo_tps[isec + 1][ilat],
01517                     clim_tropo_lats[ilat + 1],
01518                     clim_tropo_tps[isec + 1][ilat + 1], lat);
01519     return LIN(clim_tropo_secs[isec], p0, clim_tropo_secs[isec + 1], p1, sec);
01520 }
01521
01522 /*****
01523
01524 void day2doy(
01525     int year,
01526     int mon,
01527     int day,
01528     int *doy) {
01529
01530     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01531     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01532
01533     /* Get day of year... */
01534     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
01535         *doy = d0l[mon - 1] + day - 1;
01536     else
01537         *doy = d0[mon - 1] + day - 1;
01538 }
01539
01540 /*****
01541
01542 void doy2day(
01543     int year,
01544     int doy,
01545     int *mon,
01546     int *day) {
01547
01548     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01549     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01550     int i;
01551
01552     /* Get month and day... */
01553     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
01554         for (i = 11; i >= 0; i--)
01555             if (d0l[i] <= doy)
01556                 break;
01557         *mon = i + 1;
01558         *day = doy - d0l[i] + 1;
01559     } else {
01560         for (i = 11; i >= 0; i--)
01561             if (d0[i] <= doy)
01562                 break;
01563         *mon = i + 1;
01564         *day = doy - d0[i] + 1;
01565     }
01566 }
01567
01568 /*****
01569
01570 void geo2cart(
01571     double z,
01572     double lon,

```

```

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

```

```

01659
01660 void get_met_help(
01661     double t,
01662     int direct,
01663     char *metbase,
01664     double dt_met,
01665     char *filename) {
01666
01667     char repl[LEN];
01668
01669     double t6, r;
01670
01671     int year, mon, day, hour, min, sec;
01672
01673     /* Round time to fixed intervals... */
01674     if (direct == -1)
01675         t6 = floor(t / dt_met) * dt_met;
01676     else
01677         t6 = ceil(t / dt_met) * dt_met;
01678
01679     /* Decode time... */
01680     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
01681
01682     /* Set filename... */
01683     sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
01684     sprintf(repl, "%d", year);
01685     get_met_replace(filename, "YYYY", repl);
01686     sprintf(repl, "%02d", mon);
01687     get_met_replace(filename, "MM", repl);
01688     sprintf(repl, "%02d", day);
01689     get_met_replace(filename, "DD", repl);
01690     sprintf(repl, "%02d", hour);
01691     get_met_replace(filename, "HH", repl);
01692 }
01693
01694 /*****
01695
01696 void get_met_replace(
01697     char *orig,
01698     char *search,
01699     char *repl) {
01700
01701     char buffer[LEN], *ch;
01702
01703     /* Iterate... */
01704     for (int i = 0; i < 3; i++) {
01705
01706         /* Replace substring... */
01707         if (!(ch = strstr(orig, search)))
01708             return;
01709         strncpy(buffer, orig, (size_t) (ch - orig));
01710         buffer[ch - orig] = 0;
01711         sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
01712         orig[0] = 0;
01713         strcpy(orig, buffer);
01714     }
01715 }
01716
01717 /*****
01718
01719 void intpol_met_space_3d(
01720     met_t * met,
01721     float array[EX][EY][EP],
01722     double p,
01723     double lon,
01724     double lat,
01725     double *var,
01726     int *ci,
01727     double *cw,
01728     int init) {
01729
01730     /* Check longitude... */
01731     if (met->lon[met->nx - 1] > 180 && lon < 0)
01732         lon += 360;
01733
01734     /* Get interpolation indices and weights... */
01735     if (init) {
01736         ci[0] = locate_irr(met->p, met->np, p);
01737         ci[1] = locate_reg(met->lon, met->nx, lon);
01738         ci[2] = locate_reg(met->lat, met->ny, lat);
01739         cw[0] = (met->p[ci[0] + 1] - p)
01740             / (met->p[ci[0] + 1] - met->p[ci[0]]);
01741         cw[1] = (met->lon[ci[1] + 1] - lon)
01742             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01743         cw[2] = (met->lat[ci[2] + 1] - lat)
01744             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01745     }

```

```

01746
01747 /* Interpolate vertically... */
01748 double aux00 =
01749     cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
01750     + array[ci[1]][ci[2]][ci[0] + 1];
01751 double aux01 =
01752     cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
01753         array[ci[1]][ci[2] + 1][ci[0] + 1])
01754     + array[ci[1]][ci[2] + 1][ci[0] + 1];
01755 double aux10 =
01756     cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
01757         array[ci[1] + 1][ci[2]][ci[0] + 1])
01758     + array[ci[1] + 1][ci[2]][ci[0] + 1];
01759 double aux11 =
01760     cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
01761         array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
01762     + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];
01763
01764 /* Interpolate horizontally... */
01765 aux00 = cw[2] * (aux00 - aux01) + aux01;
01766 aux11 = cw[2] * (aux10 - aux11) + aux11;
01767 *var = cw[1] * (aux00 - aux11) + aux11;
01768 }
01769
01770
01771 /*****
01772
01773 void intpol_met_space_2d(
01774     met_t * met,
01775     float array[EX][EY],
01776     double lon,
01777     double lat,
01778     double *var,
01779     int *ci,
01780     double *cw,
01781     int init) {
01782
01783     /* Check longitude... */
01784     if (met->lon[met->nx - 1] > 180 && lon < 0)
01785         lon += 360;
01786
01787     /* Get interpolation indices and weights... */
01788     if (init) {
01789         ci[1] = locate_reg(met->lon, met->nx, lon);
01790         ci[2] = locate_reg(met->lat, met->ny, lat);
01791         cw[1] = (met->lon[ci[1] + 1] - lon)
01792             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01793         cw[2] = (met->lat[ci[2] + 1] - lat)
01794             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01795     }
01796
01797     /* Set variables... */
01798     double aux00 = array[ci[1]][ci[2]];
01799     double aux01 = array[ci[1]][ci[2] + 1];
01800     double aux10 = array[ci[1] + 1][ci[2]];
01801     double aux11 = array[ci[1] + 1][ci[2] + 1];
01802
01803     /* Interpolate horizontally... */
01804     if (isfinite(aux00) && isfinite(aux01))
01805         aux00 = cw[2] * (aux00 - aux01) + aux01;
01806     else if (cw[2] < 0.5)
01807         aux00 = aux01;
01808     if (isfinite(aux10) && isfinite(aux11))
01809         aux11 = cw[2] * (aux10 - aux11) + aux11;
01810     else if (cw[2] > 0.5)
01811         aux11 = aux10;
01812     if (isfinite(aux00) && isfinite(aux11))
01813         *var = cw[1] * (aux00 - aux11) + aux11;
01814     else {
01815         if (cw[1] > 0.5)
01816             *var = aux00;
01817         else
01818             *var = aux11;
01819     }
01820 }
01821
01822 /*****
01823
01824 void intpol_met_time_3d(
01825     met_t * met0,
01826     float array0[EX][EY][EP],
01827     met_t * met1,
01828     float array1[EX][EY][EP],
01829     double ts,
01830     double p,
01831     double lon,
01832     double lat,

```

```

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

```

```

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

```

```

02007      /* Increment data point counter... */
02008      if ((++atm->np) > NP)
02009          ERRMSG("Too many data points!");
02010  }
02011
02012      /* Close file... */
02013      fclose(in);
02014  }
02015
02016      /* Read binary data... */
02017      else if (ctl->atm_type == 1) {
02018
02019          /* Open file... */
02020          if (!(in = fopen(filename, "r")))
02021              return 0;
02022
02023          /* Read data... */
02024          FREAD(&atm->np, int, 1, in);
02025          FREAD(atm->time, double,
02026                (size_t) atm->np,
02027                in);
02028          FREAD(atm->p, double,
02029                (size_t) atm->np,
02030                in);
02031          FREAD(atm->lon, double,
02032                (size_t) atm->np,
02033                in);
02034          FREAD(atm->lat, double,
02035                (size_t) atm->np,
02036                in);
02037          for (iq = 0; iq < ctl->nq; iq++)
02038              FREAD(atm->q[iq], double,
02039                    (size_t) atm->np,
02040                    in);
02041
02042          /* Close file... */
02043          fclose(in);
02044      }
02045
02046      /* Read netCDF data... */
02047      else if (ctl->atm_type == 2) {
02048
02049          /* Open file... */
02050          if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02051              return 0;
02052
02053          /* Get dimensions... */
02054          NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
02055          NC(nc_inq_dimlen(ncid, dimid, &nparts));
02056          atm->np = (int) nparts;
02057          if (atm->np > NP)
02058              ERRMSG("Too many particles!");
02059
02060          /* Get time... */
02061          NC(nc_inq_varid(ncid, "time", &varid));
02062          NC(nc_get_var_double(ncid, varid, &t0));
02063          for (ip = 0; ip < atm->np; ip++)
02064              atm->time[ip] = t0;
02065
02066          /* Read geolocations... */
02067          NC(nc_inq_varid(ncid, "PRESS", &varid));
02068          NC(nc_get_var_double(ncid, varid, atm->p));
02069          NC(nc_inq_varid(ncid, "LON", &varid));
02070          NC(nc_get_var_double(ncid, varid, atm->lon));
02071          NC(nc_inq_varid(ncid, "LAT", &varid));
02072          NC(nc_get_var_double(ncid, varid, atm->lat));
02073
02074          /* Read variables... */
02075          if (ctl->qnt_p >= 0)
02076              if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
02077                  NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
02078          if (ctl->qnt_t >= 0)
02079              if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
02080                  NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
02081          if (ctl->qnt_u >= 0)
02082              if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
02083                  NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
02084          if (ctl->qnt_v >= 0)
02085              if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
02086                  NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
02087          if (ctl->qnt_w >= 0)
02088              if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
02089                  NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
02090          if (ctl->qnt_h2o >= 0)
02091              if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
02092                  NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
02093          if (ctl->qnt_o3 >= 0)

```

```

02094     if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
02095         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
02096     if (ctl->qnt_theta >= 0)
02097         if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
02098             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
02099     if (ctl->qnt_pv >= 0)
02100         if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
02101             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
02102
02103     /* Check data... */
02104     for (ip = 0; ip < atm->np; ip++)
02105         if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
02106             || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
02107             || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
02108             || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
02109             || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
02110         atm->time[ip] = GSL_NAN;
02111         atm->p[ip] = GSL_NAN;
02112         atm->lon[ip] = GSL_NAN;
02113         atm->lat[ip] = GSL_NAN;
02114         for (iq = 0; iq < ctl->nq; iq++)
02115             atm->q[iq][ip] = GSL_NAN;
02116     } else {
02117         if (ctl->qnt_h2o >= 0)
02118             atm->q[ctl->qnt_h2o][ip] *= 1.608;
02119         if (ctl->qnt_pv >= 0)
02120             atm->q[ctl->qnt_pv][ip] *= 1e6;
02121         if (atm->lon[ip] > 180)
02122             atm->lon[ip] -= 360;
02123     }
02124
02125     /* Close file... */
02126     NC(nc_close(ncid));
02127 }
02128
02129 /* Error... */
02130 else
02131     ERRMSG("Atmospheric data type not supported!");
02132
02133 /* Check number of points... */
02134 if (atm->np < 1)
02135     ERRMSG("Can not read any data!");
02136
02137 /* Return success... */
02138 return 1;
02139 }
02140
02141 /*****
02142
02143 void read_ctl(
02144     const char *filename,
02145     int argc,
02146     char *argv[],
02147     ctl_t * ctl) {
02148
02149     /* Write info... */
02150     printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
02151           "(executable: %s | compiled: %s, %s)\n\n",
02152           argv[0], __DATE__, __TIME__);
02153
02154     /* Initialize quantity indices... */
02155     ctl->qnt_ens = -1;
02156     ctl->qnt_m = -1;
02157     ctl->qnt_r = -1;
02158     ctl->qnt_rho = -1;
02159     ctl->qnt_ps = -1;
02160     ctl->qnt_pt = -1;
02161     ctl->qnt_z = -1;
02162     ctl->qnt_p = -1;
02163     ctl->qnt_t = -1;
02164     ctl->qnt_u = -1;
02165     ctl->qnt_v = -1;
02166     ctl->qnt_w = -1;
02167     ctl->qnt_h2o = -1;
02168     ctl->qnt_o3 = -1;
02169     ctl->qnt_lwc = -1;
02170     ctl->qnt_iwc = -1;
02171     ctl->qnt_pc = -1;
02172     ctl->qnt_hno3 = -1;
02173     ctl->qnt_oh = -1;
02174     ctl->qnt_rh = -1;
02175     ctl->qnt_theta = -1;
02176     ctl->qnt_vh = -1;
02177     ctl->qnt_vz = -1;
02178     ctl->qnt_pv = -1;
02179     ctl->qnt_tice = -1;
02180     ctl->qnt_tsts = -1;

```



```

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

```

```

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

```

```

02355     }
02356
02357     /* PSC analysis... */
02358     ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
02359     ctl->psc_hno3 =
02360         scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
02361
02362     /* Output of atmospheric data... */
02363     scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
atm_basename);
02364     scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
02365     ctl->atm_dt_out =
02366         scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
02367     ctl->atm_filter =
02368         (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
02369     ctl->atm_stride =
02370         (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
02371     ctl->atm_type =
02372         (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
02373
02374     /* Output of CSI data... */
02375     scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
csi_basename);
02376     ctl->csi_dt_out =
02377         scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
02378     scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->
csi_obsfile);
02379     ctl->csi_obsmin =
02380         scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
02381     ctl->csi_modmin =
02382         scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
02383     ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
02384     ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
02385     ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
02386     ctl->csi_lon0 =
02387         scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
02388     ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
02389     ctl->csi_nx =
02390         (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
02391     ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
02392     ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
02393     ctl->csi_ny =
02394         (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
02395
02396     /* Output of ensemble data... */
02397     scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
ens_basename);
02398
02399     /* Output of grid data... */
02400     scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
02401         ctl->grid_basename);
02402     scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
grid_gpfile);
02403     ctl->grid_dt_out =
02404         scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
02405     ctl->grid_sparse =
02406         (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
02407     ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
02408     ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
02409     ctl->grid_nz =
02410         (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
02411     ctl->grid_lon0 =
02412         scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
02413     ctl->grid_lon1 =
02414         scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
02415     ctl->grid_nx =
02416         (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
02417     ctl->grid_lat0 =
02418         scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
02419     ctl->grid_lat1 =
02420         scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
02421     ctl->grid_ny =
02422         (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
02423
02424     /* Output of profile data... */
02425     scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
02426         ctl->prof_basename);
02427     scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
prof_obsfile);
02428     ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
02429     ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
02430     ctl->prof_nz =
02431         (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
02432     ctl->prof_lon0 =
02433         scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
02434     ctl->prof_lon1 =
02435         scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);

```

```

02436   ctl->prof_nx =
02437       (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
02438   ctl->prof_lat0 =
02439       scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
02440   ctl->prof_lat1 =
02441       scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
02442   ctl->prof_ny =
02443       (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
02444
02445   /* Output of station data... */
02446   scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
02447           ctl->stat_basename);
02448   ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
02449   ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
02450   ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
02451 }
02452
02453 /*****
02454
02455 int read_met(
02456     ctl_t * ctl,
02457     char *filename,
02458     met_t * met) {
02459
02460     char cmd[2 * LEN], levname[LEN], tstr[10];
02461
02462     int ip, dimid, ncid, varid, year, mon, day, hour;
02463
02464     size_t np, nx, ny;
02465
02466     /* Write info... */
02467     printf("Read meteorological data: %s\n", filename);
02468
02469     /* Get time from filename... */
02470     sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
02471     year = atoi(tstr);
02472     sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
02473     mon = atoi(tstr);
02474     sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
02475     day = atoi(tstr);
02476     sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
02477     hour = atoi(tstr);
02478     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
02479
02480     /* Open netCDF file... */
02481     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
02482
02483         /* Try to stage meteo file... */
02484         if (ctl->met_stage[0] != '-') {
02485             sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
02486                     year, mon, day, hour, filename);
02487             if (system(cmd) != 0)
02488                 ERRMSG("Error while staging meteo data!");
02489         }
02490
02491         /* Try to open again... */
02492         if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
02493             WARN("File not found!");
02494             return 0;
02495         }
02496     }
02497
02498     /* Get dimensions... */
02499     NC(nc_inq_dimid(ncid, "lon", &dimid));
02500     NC(nc_inq_dimlen(ncid, dimid, &nx));
02501     if (nx < 2 || nx > EX)
02502         ERRMSG("Number of longitudes out of range!");
02503
02504     NC(nc_inq_dimid(ncid, "lat", &dimid));
02505     NC(nc_inq_dimlen(ncid, dimid, &ny));
02506     if (ny < 2 || ny > EY)
02507         ERRMSG("Number of latitudes out of range!");
02508
02509     sprintf(levname, "lev");
02510     NC(nc_inq_dimid(ncid, levname, &dimid));
02511     NC(nc_inq_dimlen(ncid, dimid, &np));
02512     if (np == 1) {
02513         sprintf(levname, "lev_2");
02514         if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
02515             sprintf(levname, "plev");
02516             nc_inq_dimid(ncid, levname, &dimid);
02517         }
02518         NC(nc_inq_dimlen(ncid, dimid, &np));
02519     }
02520     if (np < 2 || np > EP)
02521         ERRMSG("Number of levels out of range!");
02522

```

```

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

```

```

02610  /* Calculate cloud properties... */
02611  read_met_cloud(met);
02612
02613  /* Close file... */
02614  NC(nc_close(ncid));
02615
02616  /* Return success... */
02617  return 1;
02618 }
02619
02620 /*****
02621
02622 void read_met_cloud(
02623     met_t * met) {
02624
02625     int ix, iy, ip;
02626
02627     /* Loop over columns... */
02628     #pragma omp parallel for default(shared) private(ix,iy,ip)
02629     for (ix = 0; ix < met->nx; ix++)
02630         for (iy = 0; iy < met->ny; iy++) {
02631
02632             /* Init... */
02633             met->pc[ix][iy] = GSL_NAN;
02634             met->cl[ix][iy] = 0;
02635
02636             /* Loop over pressure levels... */
02637             for (ip = 0; ip < met->np - 1; ip++) {
02638
02639                 /* Check pressure... */
02640                 if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
02641                     continue;
02642
02643                 /* Get cloud top pressure ... */
02644                 if (met->iwc[ix][iy][ip] > 0 || met->lwc[ix][iy][ip] > 0)
02645                     met->pc[ix][iy] = (float) met->p[ip + 1];
02646
02647                 /* Get cloud water... */
02648                 met->cl[ix][iy] += (float)
02649                     (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1])
02650                      + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
02651                     * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
02652             }
02653         }
02654     }
02655
02656 /*****
02657
02658 void read_met_extrapolate(
02659     met_t * met) {
02660
02661     int ip, ip0, ix, iy;
02662
02663     /* Loop over columns... */
02664     #pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
02665     for (ix = 0; ix < met->nx; ix++)
02666         for (iy = 0; iy < met->ny; iy++) {
02667
02668             /* Find lowest valid data point... */
02669             for (ip0 = met->np - 1; ip0 >= 0; ip0--)
02670                 if (!isfinite(met->t[ix][iy][ip0])
02671                     || !isfinite(met->u[ix][iy][ip0])
02672                     || !isfinite(met->v[ix][iy][ip0])
02673                     || !isfinite(met->w[ix][iy][ip0]))
02674                     break;
02675
02676             /* Extrapolate... */
02677             for (ip = ip0; ip >= 0; ip--) {
02678                 met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
02679                 met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
02680                 met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
02681                 met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
02682                 met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
02683                 met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
02684                 met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
02685                 met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
02686             }
02687         }
02688     }
02689
02690 /*****
02691
02692 void read_met_geopot(
02693     met_t * met) {
02694
02695     const int dx = 6, dy = 4;
02696

```

```

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

```

```

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

```



```

02870
02871 void read_met_m12pl(
02872     ctl_t * ctl,
02873     met_t * met,
02874     float var[EX][EY][EP]) {
02875
02876     double aux[EP], p[EP], pt;
02877
02878     int ip, ip2, ix, iy;
02879
02880     /* Loop over columns... */
02881     #pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
02882     for (ix = 0; ix < met->nx; ix++)
02883         for (iy = 0; iy < met->ny; iy++) {
02884
02885             /* Copy pressure profile... */
02886             for (ip = 0; ip < met->np; ip++)
02887                 p[ip] = met->p1[ix][iy][ip];
02888
02889             /* Interpolate... */
02890             for (ip = 0; ip < ctl->met_np; ip++) {
02891                 pt = ctl->met_p[ip];
02892                 if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
02893                     pt = p[0];
02894                 else if ((pt > p[met->np - 1] && p[1] > p[0])
02895                     || (pt < p[met->np - 1] && p[1] < p[0]))
02896                     pt = p[met->np - 1];
02897                 ip2 = locate_irr(p, met->np, pt);
02898                 aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
02899                     p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
02900             }
02901
02902             /* Copy data... */
02903             for (ip = 0; ip < ctl->met_np; ip++)
02904                 var[ix][iy][ip] = (float) aux[ip];
02905         }
02906     }
02907
02908     /*****
02909
02910 void read_met_periodic(
02911     met_t * met) {
02912
02913     /* Check longitudes... */
02914     if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
02915         + met->lon[1] - met->lon[0] - 360) < 0.01))
02916         return;
02917
02918     /* Increase longitude counter... */
02919     if ((++met->nx) > EX)
02920         ERRMSG("Cannot create periodic boundary conditions!");
02921
02922     /* Set longitude... */
02923     met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
lon[0];
02924
02925     /* Loop over latitudes and pressure levels... */
02926     #pragma omp parallel for default(shared)
02927     for (int iy = 0; iy < met->ny; iy++) {
02928         met->ps[met->nx - 1][iy] = met->ps[0][iy];
02929         met->zs[met->nx - 1][iy] = met->zs[0][iy];
02930         for (int ip = 0; ip < met->np; ip++) {
02931             met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
02932             met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
02933             met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
02934             met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
02935             met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
02936             met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
02937             met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
02938             met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
02939         }
02940     }
02941 }
02942
02943 /*****
02944
02945 void read_met_pv(
02946     met_t * met) {
02947
02948     double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
02949         dtdp, dudp, dvdp, latr, vort, pows[EP];
02950
02951     int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
02952
02953     /* Set powers... */
02954     for (ip = 0; ip < met->np; ip++)
02955         pows[ip] = pow(1000. / met->p[ip], 0.286);

```

```

02956
02957 /* Loop over grid points... */
02958 #pragma omp parallel for default(shared)
    private(ix,ix0,ixl,iy,iy0,iyl,latr,dx,dy,c0,c1,cr,vort,ip,ip0,ipl,dp0,dp1,denom,dt dx,dvdx,dt dy,dudy,dt dp,dudp,dvdp)
    for (ix = 0; ix < met->nx; ix++) {
02959
02960
02961 /* Set indices... */
02962 ix0 = GSL_MAX(ix - 1, 0);
02963 ixl = GSL_MIN(ix + 1, met->nx - 1);
02964
02965 /* Loop over grid points... */
02966 for (iy = 0; iy < met->ny; iy++) {
02967
02968 /* Set indices... */
02969 iy0 = GSL_MAX(iy - 1, 0);
02970 iyl = GSL_MIN(iy + 1, met->ny - 1);
02971
02972 /* Set auxiliary variables... */
02973 latr = 0.5 * (met->lat[iyl] + met->lat[iy0]);
02974 dx = 1000. * DEG2DX(met->lon[ixl] - met->lon[ix0], latr);
02975 dy = 1000. * DEG2DY(met->lat[iyl] - met->lat[iy0]);
02976 c0 = cos(met->lat[iy0] / 180. * M_PI);
02977 c1 = cos(met->lat[iyl] / 180. * M_PI);
02978 cr = cos(latr / 180. * M_PI);
02979 vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
02980
02981 /* Loop over grid points... */
02982 for (ip = 0; ip < met->np; ip++) {
02983
02984 /* Get gradients in longitude... */
02985 dtdx = (met->t[ixl][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
02986 dvdx = (met->v[ixl][iy][ip] - met->v[ix0][iy][ip]) / dx;
02987
02988 /* Get gradients in latitude... */
02989 dtdy = (met->t[ix][iyl][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
02990 dudy = (met->u[ix][iyl][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
02991
02992 /* Set indices... */
02993 ip0 = GSL_MAX(ip - 1, 0);
02994 ipl = GSL_MIN(ip + 1, met->np - 1);
02995
02996 /* Get gradients in pressure... */
02997 dp0 = 100. * (met->p[ip] - met->p[ip0]);
02998 dp1 = 100. * (met->p[ipl] - met->p[ip]);
02999 if (ip != ip0 && ip != ipl) {
03000     denom = dp0 * dp1 * (dp0 + dp1);
03001     dtdp = (dp0 * dp0 * met->t[ix][iy][ipl] * pows[ipl]
03002            - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
03003            + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
03004            / denom;
03005     dudp = (dp0 * dp0 * met->u[ix][iy][ipl]
03006            - dp1 * dp1 * met->u[ix][iy][ip0]
03007            + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
03008            / denom;
03009     dvdp = (dp0 * dp0 * met->v[ix][iy][ipl]
03010            - dp1 * dp1 * met->v[ix][iy][ip0]
03011            + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
03012            / denom;
03013 } else {
03014     denom = dp0 + dp1;
03015     dtdp =
03016         (met->t[ix][iy][ipl] * pows[ipl] -
03017          met->t[ix][iy][ip0] * pows[ip0]) / denom;
03018     dudp = (met->u[ix][iy][ipl] - met->u[ix][iy][ip0]) / denom;
03019     dvdp = (met->v[ix][iy][ipl] - met->v[ix][iy][ip0]) / denom;
03020 }
03021
03022 /* Calculate PV... */
03023 met->pv[ix][iy][ip] = (float)
03024     (1e6 * G0 *
03025      (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
03026 }
03027 }
03028 }
03029
03030 /* Fix for polar regions... */
03031 #pragma omp parallel for default(shared) private(ix,ip)
03032 for (ix = 0; ix < met->nx; ix++)
03033     for (ip = 0; ip < met->np; ip++) {
03034         met->pv[ix][0][ip]
03035             = met->pv[ix][1][ip]
03036             = met->pv[ix][2][ip];
03037         met->pv[ix][met->ny - 1][ip]
03038             = met->pv[ix][met->ny - 2][ip]
03039             = met->pv[ix][met->ny - 3][ip];
03040     }
03041 }

```

```

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

```

```

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

```

```

03216     for (ix = 0; ix < met->nx; ix++)
03217     for (iy = 0; iy < met->ny; iy++)
03218         met->pt[ix][iy] = (float) clim_tropo(met->time, met->lat[iy]);
03219 }
03220
03221 /* Use cold point... */
03222 else if (ctl->met_tropo == 2) {
03223
03224     /* Loop over grid points... */
03225 #pragma omp parallel for default(shared) private(ix,iy,iz,t,t2)
03226     for (ix = 0; ix < met->nx; ix++)
03227         for (iy = 0; iy < met->ny; iy++) {
03228
03229             /* Interpolate temperature profile... */
03230             for (iz = 0; iz < met->np; iz++)
03231                 t[iz] = met->t[ix][iy][iz];
03232             spline(z, t, met->np, z2, t2, 171);
03233
03234             /* Find minimum... */
03235             iz = (int) gsl_stats_min_index(t2, 1, 171);
03236             if (iz > 0 && iz < 170)
03237                 met->pt[ix][iy] = (float) p2[iz];
03238             else
03239                 met->pt[ix][iy] = GSL_NAN;
03240         }
03241     }
03242
03243 /* Use WMO definition... */
03244 else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
03245
03246     /* Loop over grid points... */
03247 #pragma omp parallel for default(shared) private(ix,iy,iz,iz2,t,t2,found)
03248     for (ix = 0; ix < met->nx; ix++)
03249         for (iy = 0; iy < met->ny; iy++) {
03250
03251             /* Interpolate temperature profile... */
03252             for (iz = 0; iz < met->np; iz++)
03253                 t[iz] = met->t[ix][iy][iz];
03254             spline(z, t, met->np, z2, t2, 191);
03255
03256             /* Find 1st tropopause... */
03257             met->pt[ix][iy] = GSL_NAN;
03258             for (iz = 0; iz <= 170; iz++) {
03259                 found = 1;
03260                 for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
03261                     if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
03262                         * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
03263                         found = 0;
03264                         break;
03265                     }
03266                 if (found) {
03267                     if (iz > 0 && iz < 170)
03268                         met->pt[ix][iy] = (float) p2[iz];
03269                     break;
03270                 }
03271             }
03272
03273             /* Find 2nd tropopause... */
03274             if (ctl->met_tropo == 4) {
03275                 met->pt[ix][iy] = GSL_NAN;
03276                 for (; iz <= 170; iz++) {
03277                     found = 1;
03278                     for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
03279                         if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
03280                             * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) < 3.0) {
03281                             found = 0;
03282                             break;
03283                         }
03284                     if (found)
03285                         break;
03286                 }
03287                 for (; iz <= 170; iz++) {
03288                     found = 1;
03289                     for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
03290                         if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
03291                             * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
03292                             found = 0;
03293                             break;
03294                         }
03295                     if (found) {
03296                         if (iz > 0 && iz < 170)
03297                             met->pt[ix][iy] = (float) p2[iz];
03298                         break;
03299                     }
03300                 }
03301             }
03302         }

```

```

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

```

```

03390
03391  /* Check for missing variables... */
03392  if (!contain) {
03393      if (strlen(defvalue) > 0)
03394          sprintf(rval, "%s", defvalue);
03395      else {
03396          sprintf(msg, "Missing variable %s!\n", fullname1);
03397          ERRMSG(msg);
03398      }
03399  }
03400
03401  /* Write info... */
03402  printf("%s = %s\n", fullname1, rval);
03403
03404  /* Return values... */
03405  if (value != NULL)
03406      sprintf(value, "%s", rval);
03407  return atof(rval);
03408 }
03409
03410 /*****
03411
03412 void spline(
03413     double *x,
03414     double *y,
03415     int n,
03416     double *x2,
03417     double *y2,
03418     int n2) {
03419
03420     gsl_interp_accel *acc;
03421
03422     gsl_spline *s;
03423
03424     /* Allocate... */
03425     acc = gsl_interp_accel_alloc();
03426     s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
03427
03428     /* Interpolate temperature profile... */
03429     gsl_spline_init(s, x, y, (size_t) n);
03430     for (int i = 0; i < n2; i++)
03431         if (x2[i] <= x[0])
03432             y2[i] = y[0];
03433         else if (x2[i] >= x[n - 1])
03434             y2[i] = y[n - 1];
03435         else
03436             y2[i] = gsl_spline_eval(s, x2[i], acc);
03437
03438     /* Free... */
03439     gsl_spline_free(s);
03440     gsl_interp_accel_free(acc);
03441 }
03442
03443 /*****
03444
03445 double stddev(
03446     double *data,
03447     int n) {
03448
03449     if (n <= 0)
03450         return 0;
03451
03452     double avg = 0, rms = 0;
03453
03454     for (int i = 0; i < n; ++i)
03455         avg += data[i];
03456     avg /= n;
03457
03458     for (int i = 0; i < n; ++i)
03459         rms += SQR(data[i] - avg);
03460
03461     return sqrt(rms / (n - 1));
03462 }
03463
03464 /*****
03465
03466 void time2jsec(
03467     int year,
03468     int mon,
03469     int day,
03470     int hour,
03471     int min,
03472     int sec,
03473     double remain,
03474     double *jsec) {
03475
03476     struct tm t0, t1;

```

```

03477
03478     t0.tm_year = 100;
03479     t0.tm_mon = 0;
03480     t0.tm_mday = 1;
03481     t0.tm_hour = 0;
03482     t0.tm_min = 0;
03483     t0.tm_sec = 0;
03484
03485     t1.tm_year = year - 1900;
03486     t1.tm_mon = mon - 1;
03487     t1.tm_mday = day;
03488     t1.tm_hour = hour;
03489     t1.tm_min = min;
03490     t1.tm_sec = sec;
03491
03492     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
03493 }
03494
03495 /*****
03496
03497 void timer(
03498     const char *name,
03499     int id,
03500     int mode) {
03501
03502     static double starttime[NTIMER], runtime[NTIMER];
03503
03504     /* Check id... */
03505     if (id < 0 || id >= NTIMER)
03506         ERRMSG("Too many timers!");
03507
03508     /* Start timer... */
03509     if (mode == 1) {
03510         if (starttime[id] <= 0)
03511             starttime[id] = omp_get_wtime();
03512         else
03513             ERRMSG("Timer already started!");
03514     }
03515
03516     /* Stop timer... */
03517     else if (mode == 2) {
03518         if (starttime[id] > 0) {
03519             runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
03520             starttime[id] = -1;
03521         }
03522     }
03523
03524     /* Print timer... */
03525     else if (mode == 3) {
03526         printf("%s = %.3f s\n", name, runtime[id]);
03527         runtime[id] = 0;
03528     }
03529 }
03530
03531 /*****
03532
03533 void write_atm(
03534     const char *filename,
03535     ctl_t *ctl,
03536     atm_t *atm,
03537     double t) {
03538
03539     FILE *in, *out;
03540
03541     char line[LEN];
03542
03543     double r, t0, t1;
03544
03545     int ip, iq, year, mon, day, hour, min, sec;
03546
03547     /* Set time interval for output... */
03548     t0 = t - 0.5 * ctl->dt_mod;
03549     t1 = t + 0.5 * ctl->dt_mod;
03550
03551     /* Write info... */
03552     printf("Write atmospheric data: %s\n", filename);
03553
03554     /* Write ASCII data... */
03555     if (ctl->atm_type == 0) {
03556
03557         /* Check if gnuplot output is requested... */
03558         if (ctl->atm_gpfile[0] != '-') {
03559
03560             /* Create gnuplot pipe... */
03561             if (!(out = popen("gnuplot", "w")))
03562                 ERRMSG("Cannot create pipe to gnuplot!");
03563

```



```

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

```

```

03651     else
03652         ERRMSG("Atmospheric data type not supported!");
03653 }
03654
03655 /*****
03656
03657 void write_csi(
03658     const char *filename,
03659     ctl_t * ctl,
03660     atm_t * atm,
03661     double t) {
03662
03663     static FILE *in, *out;
03664
03665     static char line[LEN];
03666
03667     static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
03668         rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
03669
03670     static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
03671
03672     /* Init... */
03673     if (t == ctl->t_start) {
03674
03675         /* Check quantity index for mass... */
03676         if (ctl->qnt_m < 0)
03677             ERRMSG("Need quantity mass!");
03678
03679         /* Open observation data file... */
03680         printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
03681         if (!(in = fopen(ctl->csi_obsfile, "r")))
03682             ERRMSG("Cannot open file!");
03683
03684         /* Create new file... */
03685         printf("Write CSI data: %s\n", filename);
03686         if (!(out = fopen(filename, "w")))
03687             ERRMSG("Cannot create file!");
03688
03689         /* Write header... */
03690         fprintf(out,
03691             "# $1 = time [s]\n"
03692             "# $2 = number of hits (cx)\n"
03693             "# $3 = number of misses (cy)\n"
03694             "# $4 = number of false alarms (cz)\n"
03695             "# $5 = number of observations (cx + cy)\n"
03696             "# $6 = number of forecasts (cx + cz)\n"
03697             "# $7 = bias (forecasts/observations) [%]\n"
03698             "# $8 = probability of detection (POD) [%]\n"
03699             "# $9 = false alarm rate (FAR) [%]\n"
03700             "# $10 = critical success index (CSI) [%]\n\n");
03701     }
03702
03703     /* Set time interval... */
03704     t0 = t - 0.5 * ctl->dt_mod;
03705     t1 = t + 0.5 * ctl->dt_mod;
03706
03707     /* Initialize grid cells... */
03708 #pragma omp parallel for default(shared) private(ix,iy,iz)
03709     for (ix = 0; ix < ctl->csi_nx; ix++)
03710         for (iy = 0; iy < ctl->csi_ny; iy++)
03711             for (iz = 0; iz < ctl->csi_nz; iz++)
03712                 modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
03713
03714     /* Read observation data... */
03715     while (fgets(line, LEN, in)) {
03716
03717         /* Read data... */
03718         if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
03719             5)
03720             continue;
03721
03722         /* Check time... */
03723         if (rt < t0)
03724             continue;
03725         if (rt > t1)
03726             break;
03727
03728         /* Calculate indices... */
03729         ix = (int) ((rlon - ctl->csi_lon0)
03730             / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
03731         iy = (int) ((rlat - ctl->csi_lat0)
03732             / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
03733         iz = (int) ((rz - ctl->csi_z0)
03734             / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
03735
03736         /* Check indices... */
03737         if (ix < 0 || ix >= ctl->csi_nx ||

```

```

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

```

```

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

```

```

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

```

```

03999 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
04000 for (ip = 0; ip < atm->np; ip++)
04001     if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
04002
04003         /* Get index... */
04004         ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
04005         iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
04006         iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
04007
04008         /* Check indices... */
04009         if (ix < 0 || ix >= ctl->grid_nx ||
04010             iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
04011             continue;
04012
04013         /* Add mass... */
04014         if (ctl->qnt_m >= 0)
04015             mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04016         np[ix][iy][iz]++;
04017     }
04018
04019     /* Check if gnuplot output is requested... */
04020     if (ctl->grid_gpfile[0] != '-') {
04021
04022         /* Write info... */
04023         printf("Plot grid data: %s.png\n", filename);
04024
04025         /* Create gnuplot pipe... */
04026         if (!(out = popen("gnuplot", "w")))
04027             ERRMSG("Cannot create pipe to gnuplot!");
04028
04029         /* Set plot filename... */
04030         fprintf(out, "set out \"%s.png\"\n", filename);
04031
04032         /* Set time string... */
04033         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
04034         fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
04035             year, mon, day, hour, min);
04036
04037         /* Dump gnuplot file to pipe... */
04038         if (!(in = fopen(ctl->grid_gpfile, "r")))
04039             ERRMSG("Cannot open file!");
04040         while (fgets(line, LEN, in))
04041             fprintf(out, "%s", line);
04042         fclose(in);
04043     }
04044     else {
04045
04046         /* Write info... */
04047         printf("Write grid data: %s\n", filename);
04048
04049         /* Create file... */
04050         if (!(out = fopen(filename, "w")))
04051             ERRMSG("Cannot create file!");
04052     }
04053 }
04054
04055 /* Write header... */
04056 fprintf(out,
04057     "# $1 = time [s]\n"
04058     "# $2 = altitude [km]\n"
04059     "# $3 = longitude [deg]\n"
04060     "# $4 = latitude [deg]\n"
04061     "# $5 = surface area [km^2]\n"
04062     "# $6 = layer width [km]\n"
04063     "# $7 = number of particles [l]\n"
04064     "# $8 = column density [kg/m^2]\n"
04065     "# $9 = volume mixing ratio [ppv]\n\n");
04066
04067 /* Write data... */
04068 for (ix = 0; ix < ctl->grid_nx; ix++) {
04069     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
04070         fprintf(out, "\n");
04071     for (iy = 0; iy < ctl->grid_ny; iy++) {
04072         if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
04073             fprintf(out, "\n");
04074         for (iz = 0; iz < ctl->grid_nz; iz++)
04075             if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
04076
04077                 /* Set coordinates... */
04078                 z = ctl->grid_z0 + dz * (iz + 0.5);
04079                 lon = ctl->grid_lon0 + dlon * (ix + 0.5);
04080                 lat = ctl->grid_lat0 + dlat * (iy + 0.5);
04081
04082                 /* Get pressure and temperature... */
04083                 press = P(z);
04084                 intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press, lon,
04085                     lat, &temp, ci, cw, 1);

```

```

04086
04087     /* Calculate surface area... */
04088     area = dlat * dlon * SQR(RE * M_PI / 180.)
04089         * cos(lat * M_PI / 180.);
04090
04091     /* Calculate column density... */
04092     cd = mass[ix][iy][iz] / (1e6 * area);
04093
04094     /* Calculate volume mixing ratio... */
04095     rho_air = 100. * press / (RA * temp);
04096     vmr = (ctl->molmass > 0) ? MA / ctl->molmass * mass[ix][iy][iz]
04097         / (rho_air * 1e6 * area * 1e3 * dz) : GSL_NAN;
04098
04099     /* Write output... */
04100     fprintf(out, "%.2f %g %g %g %g %g %d %g %g\n",
04101         t, z, lon, lat, area, dz, np[ix][iy][iz], cd, vmr);
04102 }
04103 }
04104 }
04105
04106 /* Close file... */
04107 fclose(out);
04108 }
04109
04110 /*****
04111
04112 void write_prof(
04113     const char *filename,
04114     ctl_t * ctl,
04115     met_t * met0,
04116     met_t * met1,
04117     atm_t * atm,
04118     double t) {
04119
04120     static FILE *in, *out;
04121
04122     static char line[LEN];
04123
04124     static double mass[GX][GY][GZ], obsmean[GX][GY], rt, rz, rlon, rlat, robs,
04125         t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, vmr, h2o,
04126         o3, cw[3];
04127
04128     static int obscount[GX][GY], ip, ix, iy, iz, okay, ci[3];
04129
04130     /* Init... */
04131     if (t == ctl->t_start) {
04132
04133         /* Check quantity index for mass... */
04134         if (ctl->qnt_m < 0)
04135             ERRMSG("Need quantity mass!");
04136
04137         /* Check dimensions... */
04138         if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
04139             ERRMSG("Grid dimensions too large!");
04140
04141         /* Check molar mass... */
04142         if (ctl->molmass <= 0)
04143             ERRMSG("Specify molar mass!");
04144
04145         /* Open observation data file... */
04146         printf("Read profile observation data: %s\n", ctl->prof_obsfile);
04147         if (!(in = fopen(ctl->prof_obsfile, "r")))
04148             ERRMSG("Cannot open file!");
04149
04150         /* Create new output file... */
04151         printf("Write profile data: %s\n", filename);
04152         if (!(out = fopen(filename, "w")))
04153             ERRMSG("Cannot create file!");
04154
04155         /* Write header... */
04156         fprintf(out,
04157             "# $1 = time [s]\n"
04158             "# $2 = altitude [km]\n"
04159             "# $3 = longitude [deg]\n"
04160             "# $4 = latitude [deg]\n"
04161             "# $5 = pressure [hPa]\n"
04162             "# $6 = temperature [K]\n"
04163             "# $7 = volume mixing ratio [ppv]\n"
04164             "# $8 = H2O volume mixing ratio [ppv]\n"
04165             "# $9 = O3 volume mixing ratio [ppv]\n"
04166             "# $10 = observed BT index [K]\n");
04167
04168         /* Set grid box size... */
04169         dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
04170         dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
04171         dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
04172     }

```

```

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

```



```

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

```

```

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

```

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.

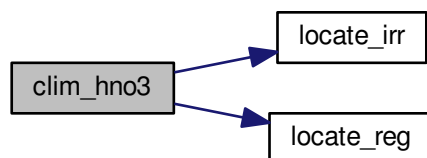
Definition at line 295 of file [libtrac.c](#).

```

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

```

Here is the call graph for this function:

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

Climatology of OH number concentrations.

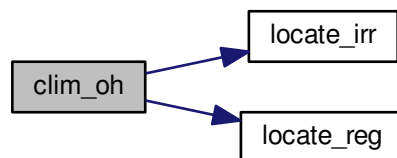
Definition at line 1322 of file [libtrac.c](#).

```

01325         {
01326
01327         /* Get seconds since begin of year... */
01328         double sec = FMOD(t, 365.25 * 86400.);
01329         while (sec < 0)
01330             sec += 365.25 * 86400.;
01331
01332         /* Check pressure... */
01333         if (p < clim_oh_ps[0])
01334             p = clim_oh_ps[0];
01335         else if (p > clim_oh_ps[33])
01336             p = clim_oh_ps[33];
01337
01338         /* Get indices... */
01339         int isec = locate_irr(clim_oh_secs, 12, sec);
01340         int ilat = locate_reg(clim_oh_lats, 18, lat);
01341         int ip = locate_irr(clim_oh_ps, 34, p);
01342
01343         /* Interpolate OH climatology (Pommrich et al., 2014)... */
01344         double aux00 = LIN(clim_oh_ps[ip],
01345                             clim_oh_var[isec][ilat][ip],
01346                             clim_oh_ps[ip + 1],
01347                             clim_oh_var[isec][ilat][ip + 1], p);
01348         double aux01 = LIN(clim_oh_ps[ip],
01349                             clim_oh_var[isec][ilat + 1][ip],
01350                             clim_oh_ps[ip + 1],
01351                             clim_oh_var[isec][ilat + 1][ip + 1], p);
01352         double aux10 = LIN(clim_oh_ps[ip],
01353                             clim_oh_var[isec + 1][ilat][ip],
01354                             clim_oh_ps[ip + 1],
01355                             clim_oh_var[isec + 1][ilat][ip + 1], p);
01356         double aux11 = LIN(clim_oh_ps[ip],
01357                             clim_oh_var[isec + 1][ilat + 1][ip],
01358                             clim_oh_ps[ip + 1],
01359                             clim_oh_var[isec + 1][ilat + 1][ip + 1], p);
01360         aux00 = LIN(clim_oh_lats[ilat], aux00, clim_oh_lats[ilat + 1], aux01, lat);
01361         aux11 = LIN(clim_oh_lats[ilat], aux10, clim_oh_lats[ilat + 1], aux11, lat);
01362         return 1e6 * LIN(clim_oh_secs[isec], aux00,
01363                         clim_oh_secs[isec + 1], aux11, sec);
01364     }

```

Here is the call graph for this function:



5.21.2.5 double clim_tropo (double t, double lat)

Climatology of tropopause pressure.

Definition at line 1497 of file libtrac.c.

```

01499         {
01500
01501         /* Get seconds since begin of year... */
01502         double sec = FMOD(t, 365.25 * 86400.);
01503         while (sec < 0)
01504             sec += 365.25 * 86400.;
01505
01506         /* Get indices... */
01507         int isec = locate_irr(clim_tropo_secs, 12, sec);

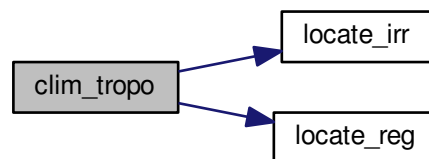
```

```

01508 int ilat = locate_reg(clim_tropo_lats, 73, lat);
01509
01510 /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
01511 double p0 = LIN(clim_tropo_lats[ilat],
01512                clim_tropo_tps[isec][ilat],
01513                clim_tropo_lats[ilat + 1],
01514                clim_tropo_tps[isec][ilat + 1], lat);
01515 double p1 = LIN(clim_tropo_lats[ilat],
01516                clim_tropo_tps[isec + 1][ilat],
01517                clim_tropo_lats[ilat + 1],
01518                clim_tropo_tps[isec + 1][ilat + 1], lat);
01519 return LIN(clim_tropo_secs[isec], p0, clim_tropo_secs[isec + 1], p1, sec);
01520 }

```

Here is the call graph for this function:



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

Get day of year from date.

Definition at line 1524 of file libtrac.c.

```

01528 {
01529
01530 int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01531 int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01532
01533 /* Get day of year... */
01534 if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
01535     *doy = d0l[mon - 1] + day - 1;
01536 else
01537     *doy = d0[mon - 1] + day - 1;
01538 }

```

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

Get date from day of year.

Definition at line 1542 of file libtrac.c.

```

01546 {
01547
01548 int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01549 int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01550 int i;
01551
01552 /* Get month and day... */
01553 if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
01554     for (i = 11; i >= 0; i--)
01555         if (d0l[i] <= doy)
01556             break;
01557     *mon = i + 1;
01558     *day = doy - d0l[i] + 1;
01559 } else {
01560     for (i = 11; i >= 0; i--)
01561         if (d0[i] <= doy)
01562             break;
01563     *mon = i + 1;
01564     *day = doy - d0[i] + 1;
01565 }
01566 }

```

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

Convert geolocation to Cartesian coordinates.

Definition at line 1570 of file libtrac.c.

```
01574         {
01575
01576     double radius = z + RE;
01577     x[0] = radius * cos(lat / 180. * M_PI) * cos(lon / 180. * M_PI);
01578     x[1] = radius * cos(lat / 180. * M_PI) * sin(lon / 180. * M_PI);
01579     x[2] = radius * sin(lat / 180. * M_PI);
01580 }
```

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

Get meteorological data for given timestep.

Definition at line 1584 of file libtrac.c.

```
01589         {
01590
01591     static int init, ip, ix, iy;
01592
01593     met_t *mets;
01594
01595     char filename[LEN];
01596
01597     /* Init... */
01598     if (t == ctl->t_start || !init) {
01599         init = 1;
01600
01601         get_met_help(t, -1, metbase, ctl->dt_met, filename);
01602         if (!read_met(ctl, filename, *met0))
01603             ERRMSG("Cannot open file!");
01604
01605         get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
dt_met, filename);
01606         if (!read_met(ctl, filename, *met1))
01607             ERRMSG("Cannot open file!");
01608 #ifdef _OPENACC
01609         met_t *met0up = *met0;
01610         met_t *met1up = *met1;
01611 #pragma acc update device(met0up[:1],met1up[:1])
01612 #endif
01613     }
01614
01615     /* Read new data for forward trajectories... */
01616     if (t > (*met1)->time && ctl->direction == 1) {
01617         mets = *met1;
01618         *met1 = *met0;
01619         *met0 = mets;
01620         get_met_help(t, 1, metbase, ctl->dt_met, filename);
01621         if (!read_met(ctl, filename, *met1))
01622             ERRMSG("Cannot open file!");
01623 #ifdef _OPENACC
01624         met_t *met1up = *met1;
01625 #pragma acc update device(met1up[:1])
01626 #endif
01627     }
01628
01629     /* Read new data for backward trajectories... */
01630     if (t < (*met0)->time && ctl->direction == -1) {
01631         mets = *met1;
01632         *met1 = *met0;
01633         *met0 = mets;
01634         get_met_help(t, -1, metbase, ctl->dt_met, filename);
01635         if (!read_met(ctl, filename, *met0))
01636             ERRMSG("Cannot open file!");
01637 #ifdef _OPENACC
01638         met_t *met0up = *met0;
01639 #pragma acc update device(met0up[:1])
01640 #endif
01641     }
01642
01643     /* Check that grids are consistent... */
```

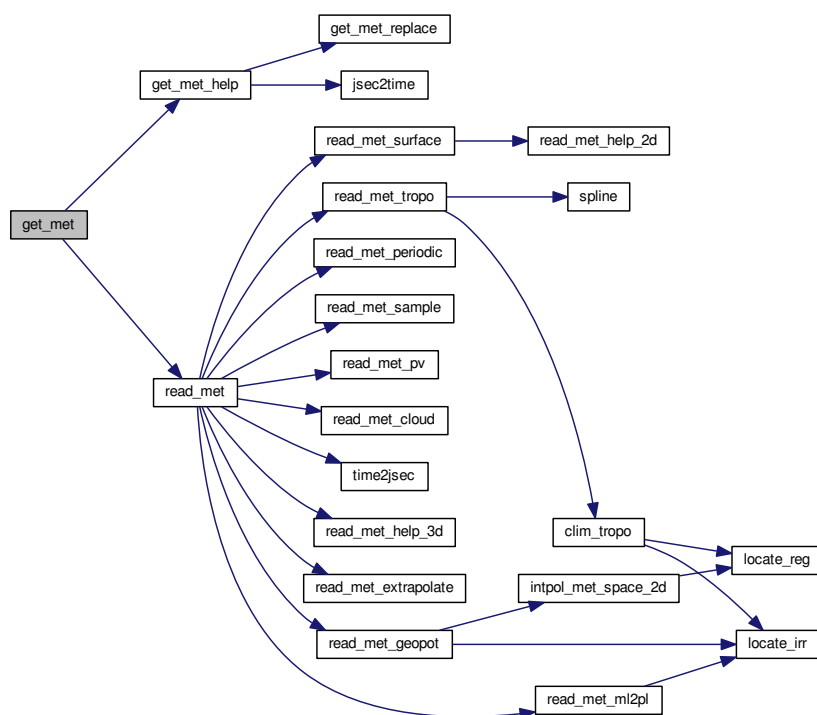


```

01644 if ((*met0)->nx != (*met1)->nx
01645 || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
01646     ERRMSG("Meteo grid dimensions do not match!");
01647 for (ix = 0; ix < (*met0)->nx; ix++)
01648     if ((*met0)->lon[ix] != (*met1)->lon[ix])
01649         ERRMSG("Meteo grid longitudes do not match!");
01650 for (iy = 0; iy < (*met0)->ny; iy++)
01651     if ((*met0)->lat[iy] != (*met1)->lat[iy])
01652         ERRMSG("Meteo grid latitudes do not match!");
01653 for (ip = 0; ip < (*met0)->np; ip++)
01654     if ((*met0)->p[ip] != (*met1)->p[ip])
01655         ERRMSG("Meteo grid pressure levels do not match!");
01656 }

```

Here is the call graph for this function:



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

Get meteorological data for timestep.

Definition at line 1660 of file libtrac.c.

```

01665 {
01666     char repl[LEN];
01667     double t6, r;
01670     int year, mon, day, hour, min, sec;
01672     /* Round time to fixed intervals... */
01673     if (direct == -1)
01674         t6 = floor(t / dt_met) * dt_met;
01675     else
01676         t6 = ceil(t / dt_met) * dt_met;
01677     t6 = ceil(t / dt_met) * dt_met;
01678 }

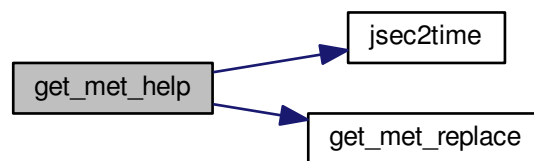
```

```

01679  /* Decode time... */
01680  jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
01681
01682  /* Set filename... */
01683  sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
01684  sprintf(repl, "%d", year);
01685  get_met_replace(filename, "YYYY", repl);
01686  sprintf(repl, "%02d", mon);
01687  get_met_replace(filename, "MM", repl);
01688  sprintf(repl, "%02d", day);
01689  get_met_replace(filename, "DD", repl);
01690  sprintf(repl, "%02d", hour);
01691  get_met_replace(filename, "HH", repl);
01692 }

```

Here is the call graph for this function:



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

Replace template strings in filename.

Definition at line 1696 of file `libtrac.c`.

```

01699      {
01700
01701      char buffer[LEN], *ch;
01702
01703      /* Iterate... */
01704      for (int i = 0; i < 3; i++) {
01705
01706          /* Replace substring... */
01707          if (!(ch = strstr(orig, search)))
01708              return;
01709          strncpy(buffer, orig, (size_t) (ch - orig));
01710          buffer[ch - orig] = 0;
01711          sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
01712          orig[0] = 0;
01713          strcpy(orig, buffer);
01714      }
01715 }

```

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

Spatial interpolation of meteorological data.

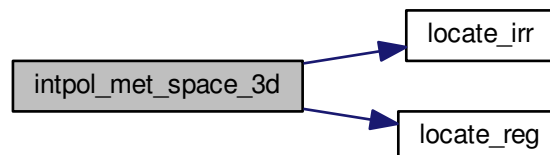
Definition at line 1719 of file `libtrac.c`.

```

01728         {
01729
01730         /* Check longitude... */
01731         if (met->lon[met->nx - 1] > 180 && lon < 0)
01732             lon += 360;
01733
01734         /* Get interpolation indices and weights... */
01735         if (init) {
01736             ci[0] = locate_irr(met->p, met->np, p);
01737             ci[1] = locate_reg(met->lon, met->nx, lon);
01738             ci[2] = locate_reg(met->lat, met->ny, lat);
01739             cw[0] = (met->p[ci[0] + 1] - p)
01740                 / (met->p[ci[0] + 1] - met->p[ci[0]]);
01741             cw[1] = (met->lon[ci[1] + 1] - lon)
01742                 / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01743             cw[2] = (met->lat[ci[2] + 1] - lat)
01744                 / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01745         }
01746
01747         /* Interpolate vertically... */
01748         double aux00 =
01749             cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
01750             + array[ci[1]][ci[2]][ci[0] + 1];
01751         double aux01 =
01752             cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
01753                 array[ci[1]][ci[2] + 1][ci[0] + 1])
01754             + array[ci[1]][ci[2] + 1][ci[0] + 1];
01755         double aux10 =
01756             cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
01757                 array[ci[1] + 1][ci[2]][ci[0] + 1])
01758             + array[ci[1] + 1][ci[2]][ci[0] + 1];
01759         double aux11 =
01760             cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
01761                 array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
01762             + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];
01763
01764         /* Interpolate horizontally... */
01765         aux00 = cw[2] * (aux00 - aux01) + aux01;
01766         aux11 = cw[2] * (aux10 - aux11) + aux11;
01767         *var = cw[1] * (aux00 - aux11) + aux11;
01768     }

```

Here is the call graph for this function:



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

Spatial interpolation of meteorological data.

Definition at line 1773 of file libtrac.c.

```

01781         {
01782
01783         /* Check longitude... */
01784         if (met->lon[met->nx - 1] > 180 && lon < 0)
01785             lon += 360;
01786

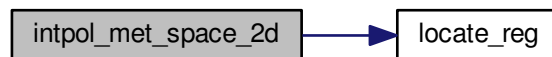
```

```

01787  /* Get interpolation indices and weights... */
01788  if (init) {
01789      ci[1] = locate_reg(met->lon, met->nx, lon);
01790      ci[2] = locate_reg(met->lat, met->ny, lat);
01791      cw[1] = (met->lon[ci[1] + 1] - lon)
01792      / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01793      cw[2] = (met->lat[ci[2] + 1] - lat)
01794      / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01795  }
01796
01797  /* Set variables... */
01798  double aux00 = array[ci[1]][ci[2]];
01799  double aux01 = array[ci[1]][ci[2] + 1];
01800  double aux10 = array[ci[1] + 1][ci[2]];
01801  double aux11 = array[ci[1] + 1][ci[2] + 1];
01802
01803  /* Interpolate horizontally... */
01804  if (isfinite(aux00) && isfinite(aux01))
01805      aux00 = cw[2] * (aux00 - aux01) + aux01;
01806  else if (cw[2] < 0.5)
01807      aux00 = aux01;
01808  if (isfinite(aux10) && isfinite(aux11))
01809      aux11 = cw[2] * (aux10 - aux11) + aux11;
01810  else if (cw[2] > 0.5)
01811      aux11 = aux10;
01812  if (isfinite(aux00) && isfinite(aux11))
01813      *var = cw[1] * (aux00 - aux11) + aux11;
01814  else {
01815      if (cw[1] > 0.5)
01816          *var = aux00;
01817      else
01818          *var = aux11;
01819  }
01820 }

```

Here is the call graph for this function:



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

Temporal interpolation of meteorological data.

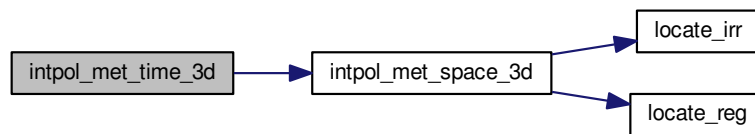
Definition at line 1824 of file `libtrac.c`.

```

01836      {
01837
01838      double var0, var1, wt;
01839
01840      /* Spatial interpolation... */
01841      intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
01842      intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, init);
01843
01844      /* Get weighting factor... */
01845      wt = (met1->time - ts) / (met1->time - met0->time);
01846
01847      /* Interpolate... */
01848      *var = wt * (var0 - var1) + var1;
01849  }

```

Here is the call graph for this function:



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

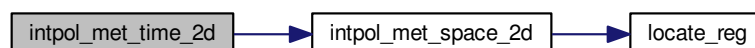
Temporal interpolation of meteorological data.

Definition at line 1853 of file [libtrac.c](#).

```

01864     {
01865
01866     double var0, var1, wt;
01867
01868     /* Spatial interpolation... */
01869     intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
01870     intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, init);
01871
01872     /* Get weighting factor... */
01873     wt = (met1->time - ts) / (met1->time - met0->time);
01874
01875     /* Interpolate... */
01876     *var = wt * (var0 - var1) + var1;
01877 }
  
```

Here is the call graph for this function:



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

Convert seconds to date.

Definition at line 1881 of file [libtrac.c](#).

```

01889     {
01890
01891     struct tm t0, *t1;
01892
01893     t0.tm_year = 100;
01894     t0.tm_mon = 0;
01895     t0.tm_mday = 1;
01896     t0.tm_hour = 0;
  
```

```

01897     t0.tm_min = 0;
01898     t0.tm_sec = 0;
01899
01900     time_t jsec0 = (time_t) jsec + timegm(&t0);
01901     t1 = gmtime(&jsec0);
01902
01903     *year = t1->tm_year + 1900;
01904     *mon = t1->tm_mon + 1;
01905     *day = t1->tm_mday;
01906     *hour = t1->tm_hour;
01907     *min = t1->tm_min;
01908     *sec = t1->tm_sec;
01909     *remain = jsec - floor(jsec);
01910 }

```

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

Find array index for irregular grid.

Definition at line 1914 of file libtrac.c.

```

01917     {
01918
01919     int ilo = 0;
01920     int ihi = n - 1;
01921     int i = (ihi + ilo) >> 1;
01922
01923     if (xx[i] < xx[i + 1])
01924         while (ihi > ilo + 1) {
01925             i = (ihi + ilo) >> 1;
01926             if (xx[i] > x)
01927                 ihi = i;
01928             else
01929                 ilo = i;
01930         } else
01931         while (ihi > ilo + 1) {
01932             i = (ihi + ilo) >> 1;
01933             if (xx[i] <= x)
01934                 ihi = i;
01935             else
01936                 ilo = i;
01937         }
01938
01939     return ilo;
01940 }

```

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

Find array index for regular grid.

Definition at line 1944 of file libtrac.c.

```

01947     {
01948
01949     /* Calculate index... */
01950     int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
01951
01952     /* Check range... */
01953     if (i < 0)
01954         i = 0;
01955     else if (i >= n - 2)
01956         i = n - 2;
01957
01958     return i;
01959 }

```

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

Read atmospheric data.

Definition at line 1963 of file libtrac.c.

```

01966         {
01967
01968     FILE *in;
01969
01970     char line[LEN], *tok;
01971
01972     double t0;
01973
01974     int dimid, ip, iq, ncid, varid;
01975
01976     size_t nparts;
01977
01978     /* Init... */
01979     atm->np = 0;
01980
01981     /* Write info... */
01982     printf("Read atmospheric data: %s\n", filename);
01983
01984     /* Read ASCII data... */
01985     if (ctl->atm_type == 0) {
01986
01987         /* Open file... */
01988         if (!(in = fopen(filename, "r"))) {
01989             WARN("File not found!");
01990             return 0;
01991         }
01992
01993         /* Read line... */
01994         while (fgets(line, LEN, in)) {
01995
01996             /* Read data... */
01997             TOK(line, tok, "%lg", atm->time[atm->np]);
01998             TOK(NULL, tok, "%lg", atm->p[atm->np]);
01999             TOK(NULL, tok, "%lg", atm->lon[atm->np]);
02000             TOK(NULL, tok, "%lg", atm->lat[atm->np]);
02001             for (iq = 0; iq < ctl->nq; iq++)
02002                 TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
02003
02004             /* Convert altitude to pressure... */
02005             atm->p[atm->np] = P(atm->p[atm->np]);
02006
02007             /* Increment data point counter... */
02008             if (++atm->np > NP)
02009                 ERRMSG("Too many data points!");
02010         }
02011
02012         /* Close file... */
02013         fclose(in);
02014     }
02015
02016     /* Read binary data... */
02017     else if (ctl->atm_type == 1) {
02018
02019         /* Open file... */
02020         if (!(in = fopen(filename, "r")))
02021             return 0;
02022
02023         /* Read data... */
02024         FREAD(&atm->np, int, 1, in);
02025         FREAD(atm->time, double,
02026             (size_t) atm->np,
02027             in);
02028         FREAD(atm->p, double,
02029             (size_t) atm->np,
02030             in);
02031         FREAD(atm->lon, double,
02032             (size_t) atm->np,
02033             in);
02034         FREAD(atm->lat, double,
02035             (size_t) atm->np,
02036             in);
02037         for (iq = 0; iq < ctl->nq; iq++)
02038             FREAD(atm->q[iq], double,
02039                 (size_t) atm->np,
02040                 in);
02041
02042         /* Close file... */

```

```

02043     fclose(in);
02044 }
02045
02046 /* Read netCDF data... */
02047 else if (ctl->atm_type == 2) {
02048
02049     /* Open file... */
02050     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02051         return 0;
02052
02053     /* Get dimensions... */
02054     NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
02055     NC(nc_inq_dimlen(ncid, dimid, &nparts));
02056     atm->np = (int) nparts;
02057     if (atm->np > NP)
02058         ERRMSG("Too many particles!");
02059
02060     /* Get time... */
02061     NC(nc_inq_varid(ncid, "time", &varid));
02062     NC(nc_get_var_double(ncid, varid, &t0));
02063     for (ip = 0; ip < atm->np; ip++)
02064         atm->time[ip] = t0;
02065
02066     /* Read geolocations... */
02067     NC(nc_inq_varid(ncid, "PRESS", &varid));
02068     NC(nc_get_var_double(ncid, varid, atm->p));
02069     NC(nc_inq_varid(ncid, "LON", &varid));
02070     NC(nc_get_var_double(ncid, varid, atm->lon));
02071     NC(nc_inq_varid(ncid, "LAT", &varid));
02072     NC(nc_get_var_double(ncid, varid, atm->lat));
02073
02074     /* Read variables... */
02075     if (ctl->qnt_p >= 0)
02076         if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
02077             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
02078     if (ctl->qnt_t >= 0)
02079         if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
02080             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
02081     if (ctl->qnt_u >= 0)
02082         if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
02083             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
02084     if (ctl->qnt_v >= 0)
02085         if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
02086             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
02087     if (ctl->qnt_w >= 0)
02088         if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
02089             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
02090     if (ctl->qnt_h2o >= 0)
02091         if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
02092             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
02093     if (ctl->qnt_o3 >= 0)
02094         if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
02095             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
02096     if (ctl->qnt_theta >= 0)
02097         if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
02098             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
02099     if (ctl->qnt_pv >= 0)
02100         if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
02101             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
02102
02103     /* Check data... */
02104     for (ip = 0; ip < atm->np; ip++)
02105         if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
02106             || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
02107             || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
02108             || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
02109             || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
02110             atm->time[ip] = GSL_NAN;
02111             atm->p[ip] = GSL_NAN;
02112             atm->lon[ip] = GSL_NAN;
02113             atm->lat[ip] = GSL_NAN;
02114             for (iq = 0; iq < ctl->nq; iq++)
02115                 atm->q[iq][ip] = GSL_NAN;
02116         } else {
02117             if (ctl->qnt_h2o >= 0)
02118                 atm->q[ctl->qnt_h2o][ip] *= 1.608;
02119             if (ctl->qnt_pv >= 0)
02120                 atm->q[ctl->qnt_pv][ip] *= 1e6;
02121             if (atm->lon[ip] > 180)
02122                 atm->lon[ip] -= 360;
02123         }
02124
02125     /* Close file... */
02126     NC(nc_close(ncid));
02127 }
02128
02129 /* Error... */

```



```

02130     else
02131         ERRMSG("Atmospheric data type not supported!");
02132
02133     /* Check number of points... */
02134     if (atm->np < 1)
02135         ERRMSG("Can not read any data!");
02136
02137     /* Return success... */
02138     return 1;
02139 }

```

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

Read control parameters.

Definition at line 2143 of file libtrac.c.

```

02147     {
02148
02149         /* Write info... */
02150         printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
02151             "(executable: %s | compiled: %s, %s)\n\n",
02152             argv[0], __DATE__, __TIME__);
02153
02154         /* Initialize quantity indices... */
02155         ctl->qnt_ens = -1;
02156         ctl->qnt_m = -1;
02157         ctl->qnt_r = -1;
02158         ctl->qnt_rho = -1;
02159         ctl->qnt_ps = -1;
02160         ctl->qnt_pt = -1;
02161         ctl->qnt_z = -1;
02162         ctl->qnt_p = -1;
02163         ctl->qnt_t = -1;
02164         ctl->qnt_u = -1;
02165         ctl->qnt_v = -1;
02166         ctl->qnt_w = -1;
02167         ctl->qnt_h2o = -1;
02168         ctl->qnt_o3 = -1;
02169         ctl->qnt_lwc = -1;
02170         ctl->qnt_iwc = -1;
02171         ctl->qnt_pc = -1;
02172         ctl->qnt_hno3 = -1;
02173         ctl->qnt_oh = -1;
02174         ctl->qnt_rh = -1;
02175         ctl->qnt_theta = -1;
02176         ctl->qnt_vh = -1;
02177         ctl->qnt_vz = -1;
02178         ctl->qnt_pv = -1;
02179         ctl->qnt_tice = -1;
02180         ctl->qnt_tsts = -1;
02181         ctl->qnt_tnat = -1;
02182         ctl->qnt_stat = -1;
02183
02184         /* Read quantities... */
02185         ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
02186         if (ctl->nq > NQ)
02187             ERRMSG("Too many quantities!");
02188         for (int iq = 0; iq < ctl->nq; iq++) {
02189
02190             /* Read quantity name and format... */
02191             scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
02192             scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
02193                 ctl->qnt_format[iq]);
02194
02195             /* Try to identify quantity... */
02196             if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
02197                 ctl->qnt_ens = iq;
02198                 sprintf(ctl->qnt_unit[iq], "-");
02199             } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
02200                 ctl->qnt_m = iq;
02201                 sprintf(ctl->qnt_unit[iq], "kg");
02202             } else if (strcmp(ctl->qnt_name[iq], "x") == 0) {
02203                 ctl->qnt_r = iq;
02204                 sprintf(ctl->qnt_unit[iq], "m");
02205             } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
02206                 ctl->qnt_rho = iq;
02207                 sprintf(ctl->qnt_unit[iq], "kg/m^3");
02208             } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
02209                 ctl->qnt_ps = iq;

```

```

02210     sprintf(ctl->qnt_unit[iq], "hPa");
02211 } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
02212     ctl->qnt_pt = iq;
02213     sprintf(ctl->qnt_unit[iq], "hPa");
02214 } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
02215     ctl->qnt_z = iq;
02216     sprintf(ctl->qnt_unit[iq], "km");
02217 } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
02218     ctl->qnt_p = iq;
02219     sprintf(ctl->qnt_unit[iq], "hPa");
02220 } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
02221     ctl->qnt_t = iq;
02222     sprintf(ctl->qnt_unit[iq], "K");
02223 } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
02224     ctl->qnt_u = iq;
02225     sprintf(ctl->qnt_unit[iq], "m/s");
02226 } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
02227     ctl->qnt_v = iq;
02228     sprintf(ctl->qnt_unit[iq], "m/s");
02229 } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
02230     ctl->qnt_w = iq;
02231     sprintf(ctl->qnt_unit[iq], "hPa/s");
02232 } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
02233     ctl->qnt_h2o = iq;
02234     sprintf(ctl->qnt_unit[iq], "ppv");
02235 } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
02236     ctl->qnt_o3 = iq;
02237     sprintf(ctl->qnt_unit[iq], "ppv");
02238 } else if (strcmp(ctl->qnt_name[iq], "lwc") == 0) {
02239     ctl->qnt_lwc = iq;
02240     sprintf(ctl->qnt_unit[iq], "kg/kg");
02241 } else if (strcmp(ctl->qnt_name[iq], "iwc") == 0) {
02242     ctl->qnt_iwc = iq;
02243     sprintf(ctl->qnt_unit[iq], "kg/kg");
02244 } else if (strcmp(ctl->qnt_name[iq], "pc") == 0) {
02245     ctl->qnt_pc = iq;
02246     sprintf(ctl->qnt_unit[iq], "hPa");
02247 } else if (strcmp(ctl->qnt_name[iq], "hno3") == 0) {
02248     ctl->qnt_hno3 = iq;
02249     sprintf(ctl->qnt_unit[iq], "ppv");
02250 } else if (strcmp(ctl->qnt_name[iq], "oh") == 0) {
02251     ctl->qnt_oh = iq;
02252     sprintf(ctl->qnt_unit[iq], "molec/cm^3");
02253 } else if (strcmp(ctl->qnt_name[iq], "rh") == 0) {
02254     ctl->qnt_rh = iq;
02255     sprintf(ctl->qnt_unit[iq], "%");
02256 } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
02257     ctl->qnt_theta = iq;
02258     sprintf(ctl->qnt_unit[iq], "K");
02259 } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
02260     ctl->qnt_vh = iq;
02261     sprintf(ctl->qnt_unit[iq], "m/s");
02262 } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
02263     ctl->qnt_vz = iq;
02264     sprintf(ctl->qnt_unit[iq], "m/s");
02265 } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
02266     ctl->qnt_pv = iq;
02267     sprintf(ctl->qnt_unit[iq], "PVU");
02268 } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
02269     ctl->qnt_tice = iq;
02270     sprintf(ctl->qnt_unit[iq], "K");
02271 } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
02272     ctl->qnt_tsts = iq;
02273     sprintf(ctl->qnt_unit[iq], "K");
02274 } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
02275     ctl->qnt_tnat = iq;
02276     sprintf(ctl->qnt_unit[iq], "K");
02277 } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
02278     ctl->qnt_stat = iq;
02279     sprintf(ctl->qnt_unit[iq], "-");
02280 } else
02281     scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
02282 }
02283
02284 /* Time steps of simulation... */
02285 ctl->direction =
02286     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
02287 if (ctl->direction != -1 && ctl->direction != 1)
02288     ERRMSG("Set DIRECTION to -1 or 1!");
02289 ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
02290 ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
02291
02292 /* Meteorological data... */
02293 ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
02294 ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
02295 ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
02296 ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);

```

```

02297 ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
02298 ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
02299 ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
02300 ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
02301 if (ctl->met_np > EP)
02302     ERRMSG("Too many levels!");
02303 for (int ip = 0; ip < ctl->met_np; ip++)
02304     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
02305 ctl->met_tropo =
02306     (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
02307 scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
02308 ctl->met_dt_out =
02309     scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
02310
02311 /* Isosurface parameters... */
02312 ctl->isosurf =
02313     (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
02314 scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
02315
02316 /* Diffusion parameters... */
02317 ctl->turb_dx_trop =
02318     scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
02319 ctl->turb_dx_strat =
02320     scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
02321 ctl->turb_dz_trop =
02322     scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
02323 ctl->turb_dz_strat =
02324     scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
02325 ctl->turb_mesox =
02326     scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
02327 ctl->turb_mesoz =
02328     scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
02329
02330 /* Species parameters... */
02331 scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
02332 if (strcmp(ctl->species, "SO2") == 0) {
02333     ctl->molmass = 64.066;
02334     ctl->oh_chem[0] = 3.3e-31; /* (JPL Publication 15-10) */
02335     ctl->oh_chem[1] = 4.3; /* (JPL Publication 15-10) */
02336     ctl->oh_chem[2] = 1.6e-12; /* (JPL Publication 15-10) */
02337     ctl->oh_chem[3] = 0.0; /* (JPL Publication 15-10) */
02338     ctl->wet_depo[0] = 2.0e-05; /* (FLEXPART v10.4) */
02339     ctl->wet_depo[1] = 0.62; /* (FLEXPART v10.4) */
02340     ctl->wet_depo[2] = 1.3e-2; /* (Sander, 2015) */
02341     ctl->wet_depo[3] = 2900.0; /* (Sander, 2015) */
02342 } else {
02343     ctl->molmass =
02344         scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
02345     ctl->tdec_trop =
02346         scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
02347     ctl->tdec_strat =
02348         scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
02349     for (int ip = 0; ip < 4; ip++)
02350         ctl->oh_chem[ip] =
02351             scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
02352     for (int ip = 0; ip < 4; ip++)
02353         ctl->wet_depo[ip] =
02354             scan_ctl(filename, argc, argv, "WET_DEPO", ip, "0", NULL);
02355 }
02356
02357 /* PSC analysis... */
02358 ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
02359 ctl->psc_hno3 =
02360     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
02361
02362 /* Output of atmospheric data... */
02363 scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
atm_basename);
02364 scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
02365 ctl->atm_dt_out =
02366     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
02367 ctl->atm_filter =
02368     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
02369 ctl->atm_stride =
02370     (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
02371 ctl->atm_type =
02372     (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
02373
02374 /* Output of CSI data... */
02375 scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
csi_basename);
02376 ctl->csi_dt_out =
02377     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
02378 scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->
csi_obsfile);
02379 ctl->csi_obsmin =
02380     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);

```

```

02381 ctl->csi_modmin =
02382     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
02383 ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
02384 ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
02385 ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
02386 ctl->csi_lon0 =
02387     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
02388 ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
02389 ctl->csi_nx =
02390     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
02391 ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
02392 ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
02393 ctl->csi_ny =
02394     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
02395
02396 /* Output of ensemble data... */
02397 scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
ens_basename);
02398
02399 /* Output of grid data... */
02400 scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
02401     ctl->grid_basename);
02402 scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
grid_gpfile);
02403 ctl->grid_dt_out =
02404     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
02405 ctl->grid_sparse =
02406     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
02407 ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
02408 ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
02409 ctl->grid_nz =
02410     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
02411 ctl->grid_lon0 =
02412     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
02413 ctl->grid_lon1 =
02414     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
02415 ctl->grid_nx =
02416     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
02417 ctl->grid_lat0 =
02418     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
02419 ctl->grid_lat1 =
02420     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
02421 ctl->grid_ny =
02422     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
02423
02424 /* Output of profile data... */
02425 scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
02426     ctl->prof_basename);
02427 scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
prof_obsfile);
02428 ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
02429 ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
02430 ctl->prof_nz =
02431     (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
02432 ctl->prof_lon0 =
02433     scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
02434 ctl->prof_lon1 =
02435     scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
02436 ctl->prof_nx =
02437     (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
02438 ctl->prof_lat0 =
02439     scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
02440 ctl->prof_lat1 =
02441     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
02442 ctl->prof_ny =
02443     (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
02444
02445 /* Output of station data... */
02446 scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
02447     ctl->stat_basename);
02448 ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
02449 ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
02450 ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
02451 }

```

Here is the call graph for this function:



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

Read meteorological data file.

Definition at line 2455 of file [libtrac.c](#).

```

02458         {
02459
02460     char cmd[2 * LEN], levname[LEN], tstr[10];
02461
02462     int ip, dimid, ncid, varid, year, mon, day, hour;
02463
02464     size_t np, nx, ny;
02465
02466     /* Write info... */
02467     printf("Read meteorological data: %s\n", filename);
02468
02469     /* Get time from filename... */
02470     sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
02471     year = atoi(tstr);
02472     sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
02473     mon = atoi(tstr);
02474     sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
02475     day = atoi(tstr);
02476     sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
02477     hour = atoi(tstr);
02478     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
02479
02480     /* Open netCDF file... */
02481     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
02482
02483         /* Try to stage meteo file... */
02484         if (ctl->met_stage[0] != '-') {
02485             sprintf(cmd, "%s %d %02d %02d %s", ctl->met_stage,
02486                 year, mon, day, hour, filename);
02487             if (system(cmd) != 0)
02488                 ERRMSG("Error while staging meteo data!");
02489         }
02490
02491         /* Try to open again... */
02492         if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
02493             WARN("File not found!");
02494             return 0;
02495         }
02496     }
02497
02498     /* Get dimensions... */
02499     NC(nc_inq_dimid(ncid, "lon", &dimid));
02500     NC(nc_inq_dimlen(ncid, dimid, &nx));
02501     if (nx < 2 || nx > EX)
02502         ERRMSG("Number of longitudes out of range!");
02503
02504     NC(nc_inq_dimid(ncid, "lat", &dimid));
02505     NC(nc_inq_dimlen(ncid, dimid, &ny));
02506     if (ny < 2 || ny > EY)
02507         ERRMSG("Number of latitudes out of range!");
02508
02509     sprintf(levname, "lev");
02510     NC(nc_inq_dimid(ncid, levname, &dimid));
02511     NC(nc_inq_dimlen(ncid, dimid, &np));
02512     if (np == 1) {
02513         sprintf(levname, "lev_2");
02514         if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
  
```

```

02515     sprintf(levname, "plev");
02516     nc_inq_dimid(ncid, levname, &dimid);
02517 }
02518 NC(nc_inq_dimlen(ncid, dimid, &np));
02519 }
02520 if (np < 2 || np > EP)
02521     ERRMSG("Number of levels out of range!");
02522
02523 /* Store dimensions... */
02524 met->np = (int) np;
02525 met->nx = (int) nx;
02526 met->ny = (int) ny;
02527
02528 /* Get horizontal grid... */
02529 NC(nc_inq_varid(ncid, "lon", &varid));
02530 NC(nc_get_var_double(ncid, varid, met->lon));
02531 NC(nc_inq_varid(ncid, "lat", &varid));
02532 NC(nc_get_var_double(ncid, varid, met->lat));
02533
02534 /* Read meteorological data... */
02535 if (!read_met_help_3d(ncid, "t", "T", met, met->t, 1.0))
02536     ERRMSG("Cannot read temperature!");
02537 if (!read_met_help_3d(ncid, "u", "U", met, met->u, 1.0))
02538     ERRMSG("Cannot read zonal wind!");
02539 if (!read_met_help_3d(ncid, "v", "V", met, met->v, 1.0))
02540     ERRMSG("Cannot read meridional wind!");
02541 if (!read_met_help_3d(ncid, "w", "W", met, met->w, 0.01f))
02542     WARN("Cannot read vertical velocity");
02543 if (!read_met_help_3d(ncid, "q", "Q", met, met->h2o, (float) (MA / MH2O)))
02544     WARN("Cannot read specific humidity!");
02545 if (!read_met_help_3d(ncid, "o3", "O3", met, met->o3, (float) (MA / MO3)))
02546     WARN("Cannot read ozone data!");
02547 if (!read_met_help_3d(ncid, "clwc", "CLWC", met, met->lwc, 1.0))
02548     WARN("Cannot read cloud liquid water content!");
02549 if (!read_met_help_3d(ncid, "ciwc", "CIWC", met, met->iwc, 1.0))
02550     WARN("Cannot read cloud ice water content!");
02551
02552 /* Meteo data on pressure levels... */
02553 if (ctl->met_np <= 0) {
02554
02555     /* Read pressure levels from file... */
02556     NC(nc_inq_varid(ncid, levname, &varid));
02557     NC(nc_get_var_double(ncid, varid, met->p));
02558     for (ip = 0; ip < met->np; ip++)
02559         met->p[ip] /= 100.;
02560
02561     /* Extrapolate data for lower boundary... */
02562     read_met_extrapolate(met);
02563 }
02564
02565 /* Meteo data on model levels... */
02566 else {
02567
02568     /* Read pressure data from file... */
02569     read_met_help_3d(ncid, "pl", "PL", met, met->p, 0.01f);
02570
02571     /* Interpolate from model levels to pressure levels... */
02572     read_met_ml2pl(ctl, met, met->t);
02573     read_met_ml2pl(ctl, met, met->u);
02574     read_met_ml2pl(ctl, met, met->v);
02575     read_met_ml2pl(ctl, met, met->w);
02576     read_met_ml2pl(ctl, met, met->h2o);
02577     read_met_ml2pl(ctl, met, met->o3);
02578     read_met_ml2pl(ctl, met, met->lwc);
02579     read_met_ml2pl(ctl, met, met->iwc);
02580
02581     /* Set pressure levels... */
02582     met->np = ctl->met_np;
02583     for (ip = 0; ip < met->np; ip++)
02584         met->p[ip] = ctl->met_p[ip];
02585 }
02586
02587 /* Check ordering of pressure levels... */
02588 for (ip = 1; ip < met->np; ip++)
02589     if (met->p[ip - 1] < met->p[ip])
02590         ERRMSG("Pressure levels must be descending!");
02591
02592 /* Read surface data... */
02593 read_met_surface(ncid, met);
02594
02595 /* Create periodic boundary conditions... */
02596 read_met_periodic(met);
02597
02598 /* Downsampling... */
02599 read_met_sample(ctl, met);
02600
02601 /* Calculate geopotential heights... */

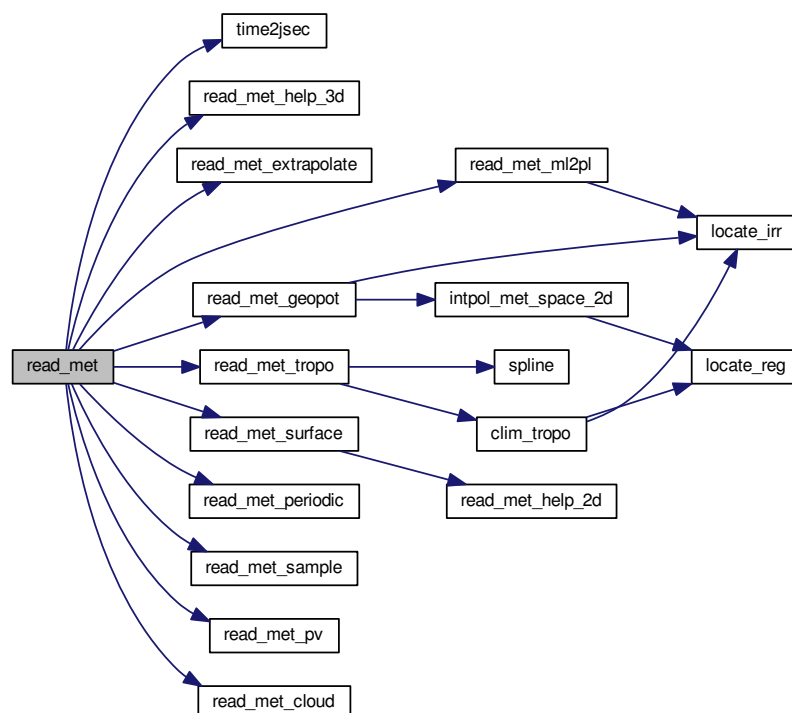
```

```

02602     read_met_geopot (met);
02603
02604     /* Calculate potential vorticity... */
02605     read_met_pv (met);
02606
02607     /* Calculate tropopause pressure... */
02608     read_met_tropo (ctl, met);
02609
02610     /* Calculate cloud properties... */
02611     read_met_cloud (met);
02612
02613     /* Close file... */
02614     NC(nc_close(ncid));
02615
02616     /* Return success... */
02617     return 1;
02618 }

```

Here is the call graph for this function:



5.21.2.22 void read_met_cloud (met_t * met)

Calculate cloud properties.

Definition at line 2622 of file libtrac.c.

```

02623     {
02624
02625     int ix, iy, ip;
02626
02627     /* Loop over columns... */
02628     #pragma omp parallel for default(shared) private(ix,iy,ip)
02629     for (ix = 0; ix < met->nx; ix++)
02630         for (iy = 0; iy < met->ny; iy++) {
02631

```

```

02632      /* Init... */
02633      met->pc[ix][iy] = GSL_NAN;
02634      met->cl[ix][iy] = 0;
02635
02636      /* Loop over pressure levels... */
02637      for (ip = 0; ip < met->np - 1; ip++) {
02638
02639          /* Check pressure... */
02640          if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
02641              continue;
02642
02643          /* Get cloud top pressure ... */
02644          if (met->iwc[ix][iy][ip] > 0 || met->lwc[ix][iy][ip] > 0)
02645              met->pc[ix][iy] = (float) met->p[ip + 1];
02646
02647          /* Get cloud water... */
02648          met->cl[ix][iy] += (float)
02649              (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1]
02650                  + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
02651               * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
02652      }
02653  }
02654 }

```

5.21.2.23 void read_met_extrapolate (met_t * met)

Extrapolate meteorological data at lower boundary.

Definition at line 2658 of file [libtrac.c](#).

```

02659      {
02660
02661          int ip, ip0, ix, iy;
02662
02663          /* Loop over columns... */
02664          #pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
02665          for (ix = 0; ix < met->nx; ix++)
02666              for (iy = 0; iy < met->ny; iy++) {
02667
02668                  /* Find lowest valid data point... */
02669                  for (ip0 = met->np - 1; ip0 >= 0; ip0--)
02670                      if (!isfinite(met->t[ix][iy][ip0])
02671                          || !isfinite(met->u[ix][iy][ip0])
02672                          || !isfinite(met->v[ix][iy][ip0])
02673                          || !isfinite(met->w[ix][iy][ip0]))
02674                      break;
02675
02676                  /* Extrapolate... */
02677                  for (ip = ip0; ip >= 0; ip--) {
02678                      met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
02679                      met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
02680                      met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
02681                      met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
02682                      met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
02683                      met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
02684                      met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
02685                      met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
02686                  }
02687              }
02688      }

```

5.21.2.24 void read_met_geopot (met_t * met)

Calculate geopotential heights.

Definition at line 2692 of file [libtrac.c](#).

```

02693      {
02694
02695          const int dx = 6, dy = 4;
02696
02697          static float help[EX][EY][EP];
02698
02699          double logp[EP], ts, z0, cw[3];

```

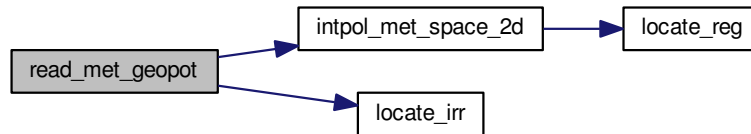


```

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

```

Here is the call graph for this function:



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

Read and convert 3D variable from meteorological data file.

Definition at line 2784 of file libtrac.c.

```

02790     {
02791
02792     float *help;
02793
02794     int ip, ix, iy, varid;
02795
02796     /* Check if variable exists... */
02797     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
02798         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
02799             return 0;
02800
02801     /* Allocate... */
02802     ALLOC(help, float, EX * EY * EP);
02803
02804     /* Read data... */
02805     NC(nc_get_var_float(ncid, varid, help));
02806
02807     /* Copy and check data... */
02808     #pragma omp parallel for default(shared) private(ix,iy,ip)
02809     for (ix = 0; ix < met->nx; ix++)
02810         for (iy = 0; iy < met->ny; iy++)
02811             for (ip = 0; ip < met->np; ip++) {
02812                 dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
02813                 if (fabsf(dest[ix][iy][ip]) < 1e14f)
02814                     dest[ix][iy][ip] *= scl;
02815                 else
02816                     dest[ix][iy][ip] = GSL_NAN;
02817             }
02818
02819     /* Free... */
02820     free(help);
02821
02822     /* Return... */
02823     return 1;
02824 }
  
```

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

Read and convert 2D variable from meteorological data file.

Definition at line 2828 of file libtrac.c.

```

02834     {
02835
02836     float *help;
02837
02838     int ix, iy, varid;
02839
  
```

```

02840  /* Check if variable exists... */
02841  if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
02842      if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
02843          return 0;
02844
02845  /* Allocate... */
02846  ALLOC(help, float, EX * EY);
02847
02848  /* Read data... */
02849  NC(nc_get_var_float(ncid, varid, help));
02850
02851  /* Copy and check data... */
02852  #pragma omp parallel for default(shared) private(ix,iy)
02853  for (ix = 0; ix < met->nx; ix++)
02854      for (iy = 0; iy < met->ny; iy++) {
02855          dest[ix][iy] = help[iy * met->nx + ix];
02856          if (fabsf(dest[ix][iy]) < 1e14f)
02857              dest[ix][iy] *= scl;
02858          else
02859              dest[ix][iy] = GSL_NAN;
02860      }
02861
02862  /* Free... */
02863  free(help);
02864
02865  /* Return... */
02866  return 1;
02867 }

```

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

Convert meteorological data from model levels to pressure levels.

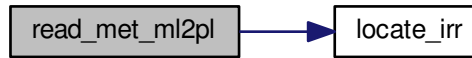
Definition at line 2871 of file libtrac.c.

```

02874      {
02875
02876      double aux[EP], p[EP], pt;
02877
02878      int ip, ip2, ix, iy;
02879
02880      /* Loop over columns... */
02881      #pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
02882      for (ix = 0; ix < met->nx; ix++)
02883          for (iy = 0; iy < met->ny; iy++) {
02884
02885              /* Copy pressure profile... */
02886              for (ip = 0; ip < met->np; ip++)
02887                  p[ip] = met->pl[ix][iy][ip];
02888
02889              /* Interpolate... */
02890              for (ip = 0; ip < ctl->met_np; ip++) {
02891                  pt = ctl->met_p[ip];
02892                  if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
02893                      pt = p[0];
02894                  else if ((pt > p[met->np - 1] && p[1] > p[0])
02895                          || (pt < p[met->np - 1] && p[1] < p[0]))
02896                      pt = p[met->np - 1];
02897                  ip2 = locate_irr(p, met->np, pt);
02898                  aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
02899                              p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
02900              }
02901
02902              /* Copy data... */
02903              for (ip = 0; ip < ctl->met_np; ip++)
02904                  var[ix][iy][ip] = (float) aux[ip];
02905          }
02906      }

```

Here is the call graph for this function:



5.21.2.28 void read_met_periodic (met_t * met)

Create meteorological data with periodic boundary conditions.

Definition at line 2910 of file libtrac.c.

```

02911         {
02912
02913         /* Check longitudes... */
02914         if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
02915                 + met->lon[1] - met->lon[0] - 360) < 0.01))
02916             return;
02917
02918         /* Increase longitude counter... */
02919         if ((++met->nx) > EX)
02920             ERRMSG("Cannot create periodic boundary conditions!");
02921
02922         /* Set longitude... */
02923         met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
lon[0];
02924
02925         /* Loop over latitudes and pressure levels... */
02926 #pragma omp parallel for default(shared)
02927         for (int iy = 0; iy < met->ny; iy++) {
02928             met->ps[met->nx - 1][iy] = met->ps[0][iy];
02929             met->zs[met->nx - 1][iy] = met->zs[0][iy];
02930             for (int ip = 0; ip < met->np; ip++) {
02931                 met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
02932                 met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
02933                 met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
02934                 met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
02935                 met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
02936                 met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
02937                 met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
02938                 met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
02939             }
02940         }
02941     }
  
```

5.21.2.29 void read_met_pv (met_t * met)

Calculate potential vorticity.

Definition at line 2945 of file libtrac.c.

```

02946         {
02947
02948         double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
02949             dtdp, dudp, dvdp, latr, vort, pows[EP];
02950
02951         int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
02952
02953         /* Set powers... */
02954         for (ip = 0; ip < met->np; ip++)
02955             pows[ip] = pow(1000. / met->p[ip], 0.286);
02956     }
  
```

```

02957  /* Loop over grid points... */
02958  #pragma omp parallel for default(shared)
02959      private(ix,ix0,ix1,iy,iy0,iy1,latr,dx,dy,c0,c1,cr,vort,ip,ip0,ip1,dp0,dp1,denom,dtdx,dvdx,dtddy,dudy,dtddp,dudp,dvdp)
02959      for (ix = 0; ix < met->nx; ix++) {
02960
02961      /* Set indices... */
02962      ix0 = GSL_MAX(ix - 1, 0);
02963      ix1 = GSL_MIN(ix + 1, met->nx - 1);
02964
02965      /* Loop over grid points... */
02966      for (iy = 0; iy < met->ny; iy++) {
02967
02968      /* Set indices... */
02969      iy0 = GSL_MAX(iy - 1, 0);
02970      iy1 = GSL_MIN(iy + 1, met->ny - 1);
02971
02972      /* Set auxiliary variables... */
02973      latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
02974      dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
02975      dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
02976      c0 = cos(met->lat[iy0] / 180. * M_PI);
02977      c1 = cos(met->lat[iy1] / 180. * M_PI);
02978      cr = cos(latr / 180. * M_PI);
02979      vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
02980
02981      /* Loop over grid points... */
02982      for (ip = 0; ip < met->np; ip++) {
02983
02984      /* Get gradients in longitude... */
02985      dtdx = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
02986      dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
02987
02988      /* Get gradients in latitude... */
02989      dtddy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
02990      dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
02991
02992      /* Set indices... */
02993      ip0 = GSL_MAX(ip - 1, 0);
02994      ip1 = GSL_MIN(ip + 1, met->np - 1);
02995
02996      /* Get gradients in pressure... */
02997      dp0 = 100. * (met->p[ip] - met->p[ip0]);
02998      dp1 = 100. * (met->p[ip1] - met->p[ip]);
02999      if (ip != ip0 && ip != ip1) {
03000          denom = dp0 * dp1 * (dp0 + dp1);
03001          dtddp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
03002                  - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
03003                  + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
03004              / denom;
03005          dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
03006                  - dp1 * dp1 * met->u[ix][iy][ip0]
03007                  + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
03008              / denom;
03009          dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
03010                  - dp1 * dp1 * met->v[ix][iy][ip0]
03011                  + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
03012              / denom;
03013      } else {
03014          denom = dp0 + dp1;
03015          dtddp =
03016              (met->t[ix][iy][ip1] * pows[ip1] -
03017               met->t[ix][iy][ip0] * pows[ip0]) / denom;
03018          dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
03019          dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
03020      }
03021
03022      /* Calculate PV... */
03023      met->pv[ix][iy][ip] = (float)
03024          (1e6 * G0 *
03025           (-dtddp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtddy));
03026      }
03027      }
03028      }
03029
03030      /* Fix for polar regions... */
03031      #pragma omp parallel for default(shared) private(ix,ip)
03032      for (ix = 0; ix < met->nx; ix++)
03033          for (ip = 0; ip < met->np; ip++) {
03034              met->pv[ix][0][ip]
03035                  = met->pv[ix][1][ip]
03036                  = met->pv[ix][2][ip];
03037              met->pv[ix][met->ny - 1][ip]
03038                  = met->pv[ix][met->ny - 2][ip]
03039                  = met->pv[ix][met->ny - 3][ip];
03040          }
03041      }

```

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

Downsampling of meteorological data.

Definition at line 3045 of file libtrac.c.

```

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

```

```

03124     }
03125 }
03126
03127 /* Downsampling... */
03128 met->nx = 0;
03129 for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
03130     met->lon[met->nx] = help->lon[ix];
03131     met->ny = 0;
03132     for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
03133         met->lat[met->ny] = help->lat[iy];
03134         met->ps[met->nx][met->ny] = help->ps[ix][iy];
03135         met->zs[met->nx][met->ny] = help->zs[ix][iy];
03136         met->np = 0;
03137         for (ip = 0; ip < help->np; ip += ctl->met_dp) {
03138             met->p[met->np] = help->p[ip];
03139             met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
03140             met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
03141             met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
03142             met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
03143             met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
03144             met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
03145             met->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];
03146             met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
03147             met->np++;
03148         }
03149         met->ny++;
03150     }
03151     met->nx++;
03152 }
03153
03154 /* Free... */
03155 free(help);
03156 }

```

5.21.2.31 void read_met_surface (int ncid, met_t * met)

Read surface data.

Definition at line 3160 of file libtrac.c.

```

03162     {
03163
03164     int ix, iy;
03165
03166     /* Read surface pressure... */
03167     if (!read_met_help_2d(ncid, "ps", "PS", met, met->ps, 0.01f)) {
03168         if (!read_met_help_2d(ncid, "lnsp", "LNSP", met, met->ps, 1.0)) {
03169             ERRMSG("Cannot not read surface pressure data!");
03170             for (ix = 0; ix < met->nx; ix++)
03171                 for (iy = 0; iy < met->ny; iy++)
03172                     met->ps[ix][iy] = (float) met->p[0];
03173         } else {
03174             for (iy = 0; iy < met->ny; iy++)
03175                 for (ix = 0; ix < met->nx; ix++)
03176                     met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
03177         }
03178     }
03179
03180     /* Read geopotential height at the surface... */
03181     if (!read_met_help_2d(ncid, "z", "Z", met, met->zs, (float) (1. / (1000. * G0))))
03182         if (!read_met_help_2d(ncid, "zm", "ZM", met, met->zs, (float) (1. / 1000.)))
03183             ERRMSG("Cannot read surface geopotential height!");
03184
03185 }
03186 }

```

Here is the call graph for this function:



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

Calculate tropopause pressure.

Definition at line 3190 of file libtrac.c.

```

03192         {
03193
03194     double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
03195           th2[200], z[EP], z2[200];
03196
03197     int found, ix, iy, iz, iz2;
03198
03199     /* Get altitude and pressure profiles... */
03200     for (iz = 0; iz < met->np; iz++)
03201         z[iz] = Z(met->p[iz]);
03202     for (iz = 0; iz <= 190; iz++) {
03203         z2[iz] = 4.5 + 0.1 * iz;
03204         p2[iz] = P(z2[iz]);
03205     }
03206
03207     /* Do not calculate tropopause... */
03208     if (ctl->met_tropo == 0)
03209         for (ix = 0; ix < met->nx; ix++)
03210             for (iy = 0; iy < met->ny; iy++)
03211                 met->pt[ix][iy] = GSL_NAN;
03212
03213     /* Use tropopause climatology... */
03214     else if (ctl->met_tropo == 1) {
03215 #pragma omp parallel for default(shared) private(ix,iy)
03216         for (ix = 0; ix < met->nx; ix++)
03217             for (iy = 0; iy < met->ny; iy++)
03218                 met->pt[ix][iy] = (float) clim_tropo(met->time, met->lat[iy]);
03219     }
03220
03221     /* Use cold point... */
03222     else if (ctl->met_tropo == 2) {
03223
03224         /* Loop over grid points... */
03225 #pragma omp parallel for default(shared) private(ix,iy,iz,t,t2)
03226         for (ix = 0; ix < met->nx; ix++)
03227             for (iy = 0; iy < met->ny; iy++) {
03228
03229                 /* Interpolate temperature profile... */
03230                 for (iz = 0; iz < met->np; iz++)
03231                     t[iz] = met->t[ix][iy][iz];
03232                 spline(z, t, met->np, z2, t2, 171);
03233
03234                 /* Find minimum... */
03235                 iz = (int) gsl_stats_min_index(t2, 1, 171);
03236                 if (iz > 0 && iz < 170)
03237                     met->pt[ix][iy] = (float) p2[iz];
03238                 else
03239                     met->pt[ix][iy] = GSL_NAN;
03240             }
03241     }
03242
03243     /* Use WMO definition... */
03244     else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
03245
03246         /* Loop over grid points... */
03247 #pragma omp parallel for default(shared) private(ix,iy,iz,iz2,t,t2,found)
03248         for (ix = 0; ix < met->nx; ix++)
03249             for (iy = 0; iy < met->ny; iy++) {
03250
03251                 /* Interpolate temperature profile... */
03252                 for (iz = 0; iz < met->np; iz++)
03253                     t[iz] = met->t[ix][iy][iz];
03254                 spline(z, t, met->np, z2, t2, 191);
03255
03256                 /* Find 1st tropopause... */
03257                 met->pt[ix][iy] = GSL_NAN;
03258                 for (iz = 0; iz <= 170; iz++) {
03259                     found = 1;
03260                     for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
03261                         if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
03262                             * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
03263                             found = 0;
03264                             break;
03265                         }
03266                     if (found) {
03267                         if (iz > 0 && iz < 170)
03268                             met->pt[ix][iy] = (float) p2[iz];

```

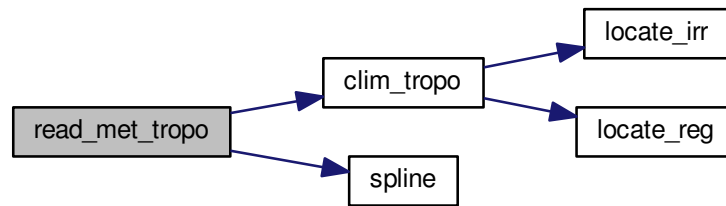


```

03269         break;
03270     }
03271 }
03272
03273 /* Find 2nd tropopause... */
03274 if (ctl->met_tropo == 4) {
03275     met->pt[ix][iy] = GSL_NAN;
03276     for (; iz <= 170; iz++) {
03277         found = 1;
03278         for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
03279             if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
03280                 * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) < 3.0) {
03281                 found = 0;
03282                 break;
03283             }
03284         if (found)
03285             break;
03286     }
03287     for (; iz <= 170; iz++) {
03288         found = 1;
03289         for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
03290             if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
03291                 * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
03292                 found = 0;
03293                 break;
03294             }
03295         if (found) {
03296             if (iz > 0 && iz < 170)
03297                 met->pt[ix][iy] = (float) p2[iz];
03298             break;
03299         }
03300     }
03301 }
03302 }
03303 }
03304
03305 /* Use dynamical tropopause... */
03306 else if (ctl->met_tropo == 5) {
03307
03308     /* Loop over grid points... */
03309     #pragma omp parallel for default(shared) private(ix,iy,iz,pv,pv2,th,th2)
03310     for (ix = 0; ix < met->nx; ix++)
03311         for (iy = 0; iy < met->ny; iy++) {
03312
03313             /* Interpolate potential vorticity profile... */
03314             for (iz = 0; iz < met->np; iz++)
03315                 pv[iz] = met->pv[ix][iy][iz];
03316             spline(z, pv, met->np, z2, pv2, 171);
03317
03318             /* Interpolate potential temperature profile... */
03319             for (iz = 0; iz < met->np; iz++)
03320                 th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
03321             spline(z, th, met->np, z2, th2, 171);
03322
03323             /* Find dynamical tropopause 3.5 PVU + 380 K */
03324             met->pt[ix][iy] = GSL_NAN;
03325             for (iz = 0; iz <= 170; iz++)
03326                 if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
03327                     if (iz > 0 && iz < 170)
03328                         met->pt[ix][iy] = (float) p2[iz];
03329                     break;
03330                 }
03331         }
03332     }
03333
03334     else
03335         ERRMSG("Cannot calculate tropopause!");
03336 }

```

Here is the call graph for this function:



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

Read a control parameter from file or command line.

Definition at line 3340 of file [libtrac.c](#).

```

03347         {
03348
03349     FILE *in = NULL;
03350
03351     char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
03352         msg[2 * LEN], rvarname[LEN], rval[LEN];
03353
03354     int contain = 0, i;
03355
03356     /* Open file... */
03357     if (filename[strlen(filename) - 1] != '-')
03358         if (!(in = fopen(filename, "r")))
03359             ERRMSG("Cannot open file!");
03360
03361     /* Set full variable name... */
03362     if (arridx >= 0) {
03363         sprintf(fullname1, "%s[%d]", varname, arridx);
03364         sprintf(fullname2, "%s[*]", varname);
03365     } else {
03366         sprintf(fullname1, "%s", varname);
03367         sprintf(fullname2, "%s", varname);
03368     }
03369
03370     /* Read data... */
03371     if (in != NULL)
03372         while (fgets(line, LEN, in))
03373             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
03374                 if (strcasecmp(rvarname, fullname1) == 0 ||
03375                     strcasecmp(rvarname, fullname2) == 0) {
03376                     contain = 1;
03377                     break;
03378                 }
03379     for (i = 1; i < argc - 1; i++)
03380         if (strcasecmp(argv[i], fullname1) == 0 ||
03381             strcasecmp(argv[i], fullname2) == 0) {
03382             sprintf(rval, "%s", argv[i + 1]);
03383             contain = 1;
03384             break;
03385         }
03386
03387     /* Close file... */
03388     if (in != NULL)
03389         fclose(in);
03390
03391     /* Check for missing variables... */
03392     if (!contain) {
03393         if (strlen(defvalue) > 0)
03394             sprintf(rval, "%s", defvalue);

```

```

03395     else {
03396         sprintf(msg, "Missing variable %s!\n", fullname1);
03397         ERRMSG(msg);
03398     }
03399 }
03400
03401 /* Write info... */
03402 printf("%s = %s\n", fullname1, rval);
03403
03404 /* Return values... */
03405 if (value != NULL)
03406     sprintf(value, "%s", rval);
03407 return atof(rval);
03408 }

```

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

Spline interpolation.

Definition at line 3412 of file libtrac.c.

```

03418     {
03419
03420     gsl_interp_accel *acc;
03421
03422     gsl_spline *s;
03423
03424     /* Allocate... */
03425     acc = gsl_interp_accel_alloc();
03426     s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
03427
03428     /* Interpolate temperature profile... */
03429     gsl_spline_init(s, x, y, (size_t) n);
03430     for (int i = 0; i < n2; i++)
03431         if (x2[i] <= x[0])
03432             y2[i] = y[0];
03433         else if (x2[i] >= x[n - 1])
03434             y2[i] = y[n - 1];
03435         else
03436             y2[i] = gsl_spline_eval(s, x2[i], acc);
03437
03438     /* Free... */
03439     gsl_spline_free(s);
03440     gsl_interp_accel_free(acc);
03441 }

```

5.21.2.35 double stddev (double * data, int n)

Calculate standard deviation.

Definition at line 3445 of file libtrac.c.

```

03447     {
03448
03449     if (n <= 0)
03450         return 0;
03451
03452     double avg = 0, rms = 0;
03453
03454     for (int i = 0; i < n; ++i)
03455         avg += data[i];
03456     avg /= n;
03457
03458     for (int i = 0; i < n; ++i)
03459         rms += SQR(data[i] - avg);
03460
03461     return sqrt(rms / (n - 1));
03462 }

```

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

Convert date to seconds.

Definition at line 3466 of file libtrac.c.

```

03474         {
03475
03476     struct tm t0, t1;
03477
03478     t0.tm_year = 100;
03479     t0.tm_mon = 0;
03480     t0.tm_mday = 1;
03481     t0.tm_hour = 0;
03482     t0.tm_min = 0;
03483     t0.tm_sec = 0;
03484
03485     t1.tm_year = year - 1900;
03486     t1.tm_mon = mon - 1;
03487     t1.tm_mday = day;
03488     t1.tm_hour = hour;
03489     t1.tm_min = min;
03490     t1.tm_sec = sec;
03491
03492     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
03493 }
```

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

Measure wall-clock time.

Definition at line 3497 of file libtrac.c.

```

03500         {
03501
03502     static double starttime[NTIMER], runtime[NTIMER];
03503
03504     /* Check id... */
03505     if (id < 0 || id >= NTIMER)
03506         ERRMSG("Too many timers!");
03507
03508     /* Start timer... */
03509     if (mode == 1) {
03510         if (starttime[id] <= 0)
03511             starttime[id] = omp_get_wtime();
03512         else
03513             ERRMSG("Timer already started!");
03514     }
03515
03516     /* Stop timer... */
03517     else if (mode == 2) {
03518         if (starttime[id] > 0) {
03519             runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
03520             starttime[id] = -1;
03521         }
03522     }
03523
03524     /* Print timer... */
03525     else if (mode == 3) {
03526         printf("%s = %.3f s\n", name, runtime[id]);
03527         runtime[id] = 0;
03528     }
03529 }
```

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

Write atmospheric data.

Definition at line 3533 of file libtrac.c.

```

03537     {
03538
03539     FILE *in, *out;
03540
03541     char line[LEN];
03542
03543     double r, t0, t1;
03544
03545     int ip, iq, year, mon, day, hour, min, sec;
03546
03547     /* Set time interval for output... */
03548     t0 = t - 0.5 * ctl->dt_mod;
03549     t1 = t + 0.5 * ctl->dt_mod;
03550
03551     /* Write info... */
03552     printf("Write atmospheric data: %s\n", filename);
03553
03554     /* Write ASCII data... */
03555     if (ctl->atm_type == 0) {
03556
03557         /* Check if gnuplot output is requested... */
03558         if (ctl->atm_gpfile[0] != '-') {
03559
03560             /* Create gnuplot pipe... */
03561             if (!(out = popen("gnuplot", "w")))
03562                 ERRMSG("Cannot create pipe to gnuplot!");
03563
03564             /* Set plot filename... */
03565             fprintf(out, "set out \"%s.png\"\n", filename);
03566
03567             /* Set time string... */
03568             jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
03569             fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
03570                     year, mon, day, hour, min);
03571
03572             /* Dump gnuplot file to pipe... */
03573             if (!(in = fopen(ctl->atm_gpfile, "r")))
03574                 ERRMSG("Cannot open file!");
03575             while (fgets(line, LEN, in))
03576                 fprintf(out, "%s", line);
03577             fclose(in);
03578         }
03579     else {
03580
03581         /* Create file... */
03582         if (!(out = fopen(filename, "w")))
03583             ERRMSG("Cannot create file!");
03584     }
03585
03586     /* Write header... */
03587     fprintf(out,
03588            "# $1 = time [s]\n"
03589            "# $2 = altitude [km]\n"
03590            "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03591     for (iq = 0; iq < ctl->nq; iq++)
03592         fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
03593                 ctl->qnt_unit[iq]);
03594     fprintf(out, "\n");
03595
03596     /* Write data... */
03597     for (ip = 0; ip < atm->np; ip += ctl->atm_stride) {
03598
03599         /* Check time... */
03600         if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
03601             continue;
03602
03603         /* Write output... */
03604         fprintf(out, "%.2f %g %g", atm->time[ip], Z(atm->p[ip]),
03605                 atm->lon[ip], atm->lat[ip]);
03606         for (iq = 0; iq < ctl->nq; iq++) {
03607             fprintf(out, " ");
03608             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
03609         }
03610         fprintf(out, "\n");
03611     }
03612 }
03613

```

```

03614     /* Close file... */
03615     fclose(out);
03616 }
03617
03618 /* Write binary data... */
03619 else if (ctl->atm_type == 1) {
03620
03621     /* Create file... */
03622     if (!(out = fopen(filename, "w")))
03623         ERRMSG("Cannot create file!");
03624
03625     /* Write data... */
03626     FWRITE(&atm->np, int,
03627           1,
03628           out);
03629     FWRITE(atm->time, double,
03630           (size_t) atm->np,
03631           out);
03632     FWRITE(atm->p, double,
03633           (size_t) atm->np,
03634           out);
03635     FWRITE(atm->lon, double,
03636           (size_t) atm->np,
03637           out);
03638     FWRITE(atm->lat, double,
03639           (size_t) atm->np,
03640           out);
03641     for (iq = 0; iq < ctl->nq; iq++)
03642         FWRITE(atm->q[iq], double,
03643               (size_t) atm->np,
03644               out);
03645
03646     /* Close file... */
03647     fclose(out);
03648 }
03649
03650 /* Error... */
03651 else
03652     ERRMSG("Atmospheric data type not supported!");
03653 }

```

Here is the call graph for this function:



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

Write CSI data.

Definition at line 3657 of file libtrac.c.

```

03661     {
03662
03663     static FILE *in, *out;
03664
03665     static char line[LEN];
03666
03667     static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
03668           rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
03669
03670     static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
03671
03672     /* Init... */
03673     if (t == ctl->t_start) {

```

```

03674
03675 /* Check quantity index for mass... */
03676 if (ctl->qnt_m < 0)
03677     ERRMSG("Need quantity mass!");
03678
03679 /* Open observation data file... */
03680 printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
03681 if (!(in = fopen(ctl->csi_obsfile, "r")))
03682     ERRMSG("Cannot open file!");
03683
03684 /* Create new file... */
03685 printf("Write CSI data: %s\n", filename);
03686 if (!(out = fopen(filename, "w")))
03687     ERRMSG("Cannot create file!");
03688
03689 /* Write header... */
03690 fprintf(out,
03691     "# $1 = time [s]\n"
03692     "# $2 = number of hits (cx)\n"
03693     "# $3 = number of misses (cy)\n"
03694     "# $4 = number of false alarms (cz)\n"
03695     "# $5 = number of observations (cx + cy)\n"
03696     "# $6 = number of forecasts (cx + cz)\n"
03697     "# $7 = bias (forecasts/observations) [%%]\n"
03698     "# $8 = probability of detection (POD) [%%]\n"
03699     "# $9 = false alarm rate (FAR) [%%]\n"
03700     "# $10 = critical success index (CSI) [%%]\n\n");
03701 }
03702
03703 /* Set time interval... */
03704 t0 = t - 0.5 * ctl->dt_mod;
03705 t1 = t + 0.5 * ctl->dt_mod;
03706
03707 /* Initialize grid cells... */
03708 #pragma omp parallel for default(shared) private(ix,iy,iz)
03709 for (ix = 0; ix < ctl->csi_nx; ix++)
03710     for (iy = 0; iy < ctl->csi_ny; iy++)
03711         for (iz = 0; iz < ctl->csi_nz; iz++)
03712             modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
03713
03714 /* Read observation data... */
03715 while (fgets(line, LEN, in)) {
03716
03717     /* Read data... */
03718     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
03719         5)
03720         continue;
03721
03722     /* Check time... */
03723     if (rt < t0)
03724         continue;
03725     if (rt > t1)
03726         break;
03727
03728     /* Calculate indices... */
03729     ix = (int) ((rlon - ctl->csi_lon0)
03730         / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
03731     iy = (int) ((rlat - ctl->csi_lat0)
03732         / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
03733     iz = (int) ((rz - ctl->csi_z0)
03734         / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
03735
03736     /* Check indices... */
03737     if (ix < 0 || ix >= ctl->csi_nx ||
03738         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
03739         continue;
03740
03741     /* Get mean observation index... */
03742     obsmean[ix][iy][iz] += robs;
03743     obscount[ix][iy][iz]++;
03744 }
03745
03746 /* Analyze model data... */
03747 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
03748 for (ip = 0; ip < atm->np; ip++) {
03749
03750     /* Check time... */
03751     if (atm->time[ip] < t0 || atm->time[ip] > t1)
03752         continue;
03753
03754     /* Get indices... */
03755     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
03756         / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
03757     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
03758         / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
03759     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
03760         / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);

```

```

03761
03762     /* Check indices... */
03763     if (ix < 0 || ix >= ctl->csi_nx ||
03764         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
03765         continue;
03766
03767     /* Get total mass in grid cell... */
03768     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
03769 }
03770
03771 /* Analyze all grid cells... */
03772 #pragma omp parallel for default(shared) private(ix,iy,iz,dlon,dlat,lat,area)
03773 for (ix = 0; ix < ctl->csi_nx; ix++)
03774     for (iy = 0; iy < ctl->csi_ny; iy++)
03775         for (iz = 0; iz < ctl->csi_nz; iz++) {
03776
03777             /* Calculate mean observation index... */
03778             if (obscount[ix][iy][iz] > 0)
03779                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
03780
03781             /* Calculate column density... */
03782             if (modmean[ix][iy][iz] > 0) {
03783                 dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
03784                 dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
03785                 lat = ctl->csi_lat0 + dlat * (iy + 0.5);
03786                 area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
03787                     * cos(lat * M_PI / 180.);
03788                 modmean[ix][iy][iz] /= (1e6 * area);
03789             }
03790
03791             /* Calculate CSI... */
03792             if (obscount[ix][iy][iz] > 0) {
03793                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
03794                     modmean[ix][iy][iz] >= ctl->csi_modmin)
03795                     cx++;
03796                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
03797                     modmean[ix][iy][iz] < ctl->csi_modmin)
03798                     cy++;
03799                 else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
03800                     modmean[ix][iy][iz] >= ctl->csi_modmin)
03801                     cz++;
03802             }
03803         }
03804
03805     /* Write output... */
03806     if (fmod(t, ctl->csi_dt_out) == 0) {
03807
03808         /* Write... */
03809         fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
03810             t, cx, cy, cz, cx + cy, cx + cz,
03811             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
03812             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
03813             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
03814             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
03815
03816         /* Set counters to zero... */
03817         cx = cy = cz = 0;
03818     }
03819
03820     /* Close file... */
03821     if (t == ctl->t_stop)
03822         fclose(out);
03823 }

```

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

Write ensemble data.

Definition at line 3827 of file libtrac.c.

```

03831     {
03832
03833         static FILE *out;
03834
03835         static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
03836             t0, t1, x[NENS][3], xm[3];
03837
03838         static int ip, iq;
03839
03840         static size_t i, n;

```



```

03841
03842  /* Init... */
03843  if (t == ctl->t_start) {
03844
03845      /* Check quantities... */
03846      if (ctl->qnt_ens < 0)
03847          ERRMSG("Missing ensemble IDs!");
03848
03849      /* Create new file... */
03850      printf("Write ensemble data: %s\n", filename);
03851      if (!(out = fopen(filename, "w")))
03852          ERRMSG("Cannot create file!");
03853
03854      /* Write header... */
03855      fprintf(out,
03856              "# $1 = time [s]\n"
03857              "# $2 = altitude [km]\n"
03858              "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03859      for (iq = 0; iq < ctl->nq; iq++)
03860          fprintf(out, "# $d = %s (mean) [%s]\n", 5 + iq,
03861                  ctl->qnt_name[iq], ctl->qnt_unit[iq]);
03862      for (iq = 0; iq < ctl->nq; iq++)
03863          fprintf(out, "# $d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
03864                  ctl->qnt_name[iq], ctl->qnt_unit[iq]);
03865      fprintf(out, "# $d = number of members\n\n", 5 + 2 * ctl->nq);
03866  }
03867
03868  /* Set time interval... */
03869  t0 = t - 0.5 * ctl->dt_mod;
03870  t1 = t + 0.5 * ctl->dt_mod;
03871
03872  /* Init... */
03873  ens = GSL_NAN;
03874  n = 0;
03875
03876  /* Loop over air parcels... */
03877  for (ip = 0; ip < atm->np; ip++) {
03878
03879      /* Check time... */
03880      if (atm->time[ip] < t0 || atm->time[ip] > t1)
03881          continue;
03882
03883      /* Check ensemble id... */
03884      if (atm->q[ctl->qnt_ens][ip] != ens) {
03885
03886          /* Write results... */
03887          if (n > 0) {
03888
03889              /* Get mean position... */
03890              xm[0] = xm[1] = xm[2] = 0;
03891              for (i = 0; i < n; i++) {
03892                  xm[0] += x[i][0] / (double) n;
03893                  xm[1] += x[i][1] / (double) n;
03894                  xm[2] += x[i][2] / (double) n;
03895              }
03896              cart2geo(xm, &dummy, &lon, &lat);
03897              fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
03898                      lat);
03899
03900              /* Get quantity statistics... */
03901              for (iq = 0; iq < ctl->nq; iq++) {
03902                  fprintf(out, " ");
03903                  fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
03904              }
03905              for (iq = 0; iq < ctl->nq; iq++) {
03906                  fprintf(out, " ");
03907                  fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
03908              }
03909              fprintf(out, " %lu\n", n);
03910          }
03911
03912          /* Init new ensemble... */
03913          ens = atm->q[ctl->qnt_ens][ip];
03914          n = 0;
03915      }
03916
03917      /* Save data... */
03918      p[n] = atm->p[ip];
03919      geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
03920      for (iq = 0; iq < ctl->nq; iq++)
03921          q[iq][n] = atm->q[iq][ip];
03922      if ((++n) >= NENS)
03923          ERRMSG("Too many data points!");
03924  }
03925
03926  /* Write results... */
03927  if (n > 0) {

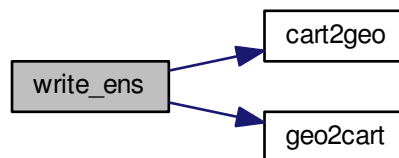
```

```

03928
03929     /* Get mean position... */
03930     xm[0] = xm[1] = xm[2] = 0;
03931     for (i = 0; i < n; i++) {
03932         xm[0] += x[i][0] / (double) n;
03933         xm[1] += x[i][1] / (double) n;
03934         xm[2] += x[i][2] / (double) n;
03935     }
03936     cart2geo(xm, &dummy, &lon, &lat);
03937     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
03938
03939     /* Get quantity statistics... */
03940     for (iq = 0; iq < ctl->nq; iq++) {
03941         fprintf(out, " ");
03942         fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
03943     }
03944     for (iq = 0; iq < ctl->nq; iq++) {
03945         fprintf(out, " ");
03946         fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
03947     }
03948     fprintf(out, " %lu\n", n);
03949 }
03950
03951 /* Close file... */
03952 if (t == ctl->t_stop)
03953     fclose(out);
03954 }

```

Here is the call graph for this function:



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

Write gridded data.

Definition at line 3958 of file `libtrac.c`.

```

03964     {
03965
03966     FILE *in, *out;
03967
03968     char line[LEN];
03969
03970     static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
03971         area, rho_air, press, temp, cd, vmr, t0, t1, r, cw[3];
03972
03973     static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec,
03974         ci[3];
03975
03976     /* Check dimensions... */
03977     if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
03978         ERRMSG("Grid dimensions too large!");
03979
03980     /* Set time interval for output... */
03981     t0 = t - 0.5 * ctl->dt_mod;
03982     t1 = t + 0.5 * ctl->dt_mod;
03983
03984     /* Set grid box size... */
03985     dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;

```

```

03986     dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
03987     dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
03988
03989     /* Initialize grid... */
03990 #pragma omp parallel for default(shared) private(ix,iy,iz)
03991     for (ix = 0; ix < ctl->grid_nx; ix++)
03992         for (iy = 0; iy < ctl->grid_ny; iy++)
03993             for (iz = 0; iz < ctl->grid_nz; iz++) {
03994                 mass[ix][iy][iz] = 0;
03995                 np[ix][iy][iz] = 0;
03996             }
03997
03998     /* Average data... */
03999 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
04000     for (ip = 0; ip < atm->np; ip++)
04001         if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
04002
04003             /* Get index... */
04004             ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
04005             iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
04006             iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
04007
04008             /* Check indices... */
04009             if (ix < 0 || ix >= ctl->grid_nx ||
04010                 iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
04011                 continue;
04012
04013             /* Add mass... */
04014             if (ctl->qnt_m >= 0)
04015                 mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04016             np[ix][iy][iz]++;
04017         }
04018
04019     /* Check if gnuplot output is requested... */
04020     if (ctl->grid_gpfile[0] != '-') {
04021
04022         /* Write info... */
04023         printf("Plot grid data: %s.png\n", filename);
04024
04025         /* Create gnuplot pipe... */
04026         if (!(out = popen("gnuplot", "w")))
04027             ERRMSG("Cannot create pipe to gnuplot!");
04028
04029         /* Set plot filename... */
04030         fprintf(out, "set out \"%s.png\"\\n", filename);
04031
04032         /* Set time string... */
04033         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
04034         fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\\n",
04035             year, mon, day, hour, min);
04036
04037         /* Dump gnuplot file to pipe... */
04038         if (!(in = fopen(ctl->grid_gpfile, "r")))
04039             ERRMSG("Cannot open file!");
04040         while (fgets(line, LEN, in))
04041             fprintf(out, "%s", line);
04042         fclose(in);
04043     }
04044     else {
04045
04046         /* Write info... */
04047         printf("Write grid data: %s\\n", filename);
04048
04049         /* Create file... */
04050         if (!(out = fopen(filename, "w")))
04051             ERRMSG("Cannot create file!");
04052     }
04053
04054     /* Write header... */
04055     fprintf(out,
04056         "# $1 = time [s]\\n"
04057         "# $2 = altitude [km]\\n"
04058         "# $3 = longitude [deg]\\n"
04059         "# $4 = latitude [deg]\\n"
04060         "# $5 = surface area [km^2]\\n"
04061         "# $6 = layer width [km]\\n"
04062         "# $7 = number of particles [l]\\n"
04063         "# $8 = column density [kg/m^2]\\n"
04064         "# $9 = volume mixing ratio [ppv]\\n\\n");
04065
04066     /* Write data... */
04067     for (ix = 0; ix < ctl->grid_nx; ix++) {
04068         if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
04069             fprintf(out, "\\n");
04070         for (iy = 0; iy < ctl->grid_ny; iy++) {
04071             if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)

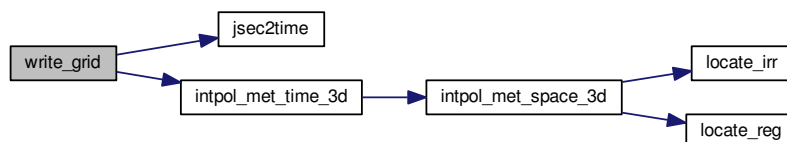
```

```

04073     fprintf(out, "\n");
04074     for (iz = 0; iz < ctl->grid_nz; iz++)
04075     if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
04076
04077         /* Set coordinates... */
04078         z = ctl->grid_z0 + dz * (iz + 0.5);
04079         lon = ctl->grid_lon0 + dlon * (ix + 0.5);
04080         lat = ctl->grid_lat0 + dlat * (iy + 0.5);
04081
04082         /* Get pressure and temperature... */
04083         press = P(z);
04084         intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press, lon,
04085             lat, &temp, ci, cw, 1);
04086
04087         /* Calculate surface area... */
04088         area = dlat * dlon * SQR(RE * M_PI / 180.)
04089             * cos(lat * M_PI / 180.);
04090
04091         /* Calculate column density... */
04092         cd = mass[ix][iy][iz] / (1e6 * area);
04093
04094         /* Calculate volume mixing ratio... */
04095         rho_air = 100. * press / (RA * temp);
04096         vmr = (ctl->molmass > 0) ? MA / ctl->molmass * mass[ix][iy][iz]
04097             / (rho_air * 1e6 * area * 1e3 * dz) : GSL_NAN;
04098
04099         /* Write output... */
04100         fprintf(out, "%.2f %g %g %g %g %g %d %g %g\n",
04101             t, z, lon, lat, area, dz, np[ix][iy][iz], cd, vmr);
04102     }
04103 }
04104 }
04105
04106 /* Close file... */
04107 fclose(out);
04108 }

```

Here is the call graph for this function:



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

Write profile data.

Definition at line 4112 of file libtrac.c.

```

04118     {
04119
04120     static FILE *in, *out;
04121
04122     static char line[LEN];
04123
04124     static double mass[GX][GY][GZ], obsmean[GX][GY], rt, rz, rlon, rlat, robs,
04125         t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, vmr, h2o,
04126         o3, cw[3];
04127
04128     static int obscount[GX][GY], ip, ix, iy, iz, okay, ci[3];
04129
04130     /* Init... */
04131     if (t == ctl->t_start) {
04132
04133         /* Check quantity index for mass... */
04134         if (ctl->qnt_m < 0)

```

```

04135     ERRMSG("Need quantity mass!");
04136
04137     /* Check dimensions... */
04138     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
04139         ERRMSG("Grid dimensions too large!");
04140
04141     /* Check molar mass... */
04142     if (ctl->molmass <= 0)
04143         ERRMSG("Specify molar mass!");
04144
04145     /* Open observation data file... */
04146     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
04147     if (!(in = fopen(ctl->prof_obsfile, "r")))
04148         ERRMSG("Cannot open file!");
04149
04150     /* Create new output file... */
04151     printf("Write profile data: %s\n", filename);
04152     if (!(out = fopen(filename, "w")))
04153         ERRMSG("Cannot create file!");
04154
04155     /* Write header... */
04156     fprintf(out,
04157         "# $1 = time [s]\n"
04158         "# $2 = altitude [km]\n"
04159         "# $3 = longitude [deg]\n"
04160         "# $4 = latitude [deg]\n"
04161         "# $5 = pressure [hPa]\n"
04162         "# $6 = temperature [K]\n"
04163         "# $7 = volume mixing ratio [ppv]\n"
04164         "# $8 = H2O volume mixing ratio [ppv]\n"
04165         "# $9 = O3 volume mixing ratio [ppv]\n"
04166         "# $10 = observed BT index [K]\n");
04167
04168     /* Set grid box size... */
04169     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
04170     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
04171     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
04172 }
04173
04174 /* Set time interval... */
04175 t0 = t - 0.5 * ctl->dt_mod;
04176 t1 = t + 0.5 * ctl->dt_mod;
04177
04178 /* Initialize... */
04179 #pragma omp parallel for default(shared) private(ix,iy,iz)
04180 for (ix = 0; ix < ctl->prof_nx; ix++)
04181     for (iy = 0; iy < ctl->prof_ny; iy++) {
04182         obsmean[ix][iy] = 0;
04183         obscount[ix][iy] = 0;
04184         for (iz = 0; iz < ctl->prof_nz; iz++)
04185             mass[ix][iy][iz] = 0;
04186     }
04187
04188 /* Read observation data... */
04189 while (fgets(line, LEN, in)) {
04190
04191     /* Read data... */
04192     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &robs) !=
04193         5)
04194         continue;
04195
04196     /* Check time... */
04197     if (rt < t0)
04198         continue;
04199     if (rt > t1)
04200         break;
04201
04202     /* Calculate indices... */
04203     ix = (int) ((rln - ctl->prof_lon0) / dlon);
04204     iy = (int) ((rln - ctl->prof_lat0) / dlat);
04205
04206     /* Check indices... */
04207     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
04208         continue;
04209
04210     /* Get mean observation index... */
04211     obsmean[ix][iy] += robs;
04212     obscount[ix][iy]++;
04213 }
04214
04215 /* Analyze model data... */
04216 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
04217 for (ip = 0; ip < atm->np; ip++) {
04218
04219     /* Check time... */
04220     if (atm->time[ip] < t0 || atm->time[ip] > t1)
04221         continue;

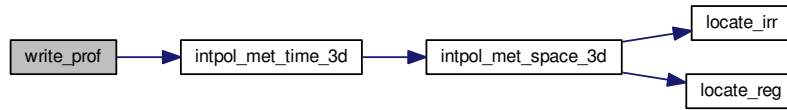
```

```

04222
04223     /* Get indices... */
04224     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
04225     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
04226     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
04227
04228     /* Check indices... */
04229     if (ix < 0 || ix >= ctl->prof_nx ||
04230         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
04231         continue;
04232
04233     /* Get total mass in grid cell... */
04234     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04235 }
04236
04237 /* Extract profiles... */
04238 for (ix = 0; ix < ctl->prof_nx; ix++)
04239     for (iy = 0; iy < ctl->prof_ny; iy++)
04240         if (obscount[ix][iy] > 0) {
04241
04242             /* Check profile... */
04243             okay = 0;
04244             for (iz = 0; iz < ctl->prof_nz; iz++)
04245                 if (mass[ix][iy][iz] > 0) {
04246                     okay = 1;
04247                     break;
04248                 }
04249             if (!okay)
04250                 continue;
04251
04252             /* Write output... */
04253             fprintf(out, "\n");
04254
04255             /* Loop over altitudes... */
04256             for (iz = 0; iz < ctl->prof_nz; iz++) {
04257
04258                 /* Set coordinates... */
04259                 z = ctl->prof_z0 + dz * (iz + 0.5);
04260                 lon = ctl->prof_lon0 + dlon * (ix + 0.5);
04261                 lat = ctl->prof_lat0 + dlat * (iy + 0.5);
04262
04263                 /* Get pressure and temperature... */
04264                 press = P(z);
04265                 intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press, lon,
04266                                     lat, &temp, ci, cw, 1);
04267                 intpol_met_time_3d(met0, met0->h2o, met1, met1->
04268                                     h2o, t, press, lon,
04269                                     lat, &h2o, ci, cw, 0);
04270                 intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press, lon,
04271                                     lat, &o3, ci, cw, 0);
04272
04273                 /* Calculate surface area... */
04274                 area = dlat * dlon * SQR(M_PI * RE / 180.)
04275                     * cos(lat * M_PI / 180.);
04276
04277                 /* Calculate volume mixing ratio... */
04278                 rho_air = 100. * press / (RA * temp);
04279                 vmr = MA / ctl->molmass * mass[ix][iy][iz]
04280                     / (rho_air * area * dz * 1e9);
04281
04282                 /* Write output... */
04283                 fprintf(out, "%.2f %g %g %g %g %g %g %g\n",
04284                         t, z, lon, lat, press, temp, vmr, h2o, o3,
04285                         obsmean[ix][iy] / obscount[ix][iy]);
04286             }
04287
04288             /* Close file... */
04289             if (t == ctl->t_stop)
04290                 fclose(out);
04291 }

```

Here is the call graph for this function:



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

Write station data.

Definition at line 4295 of file libtrac.c.

```

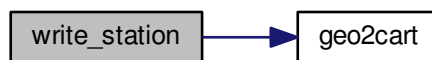
04299     {
04300
04301     static FILE *out;
04302
04303     static double rmax2, t0, t1, x0[3], x1[3];
04304
04305     /* Init... */
04306     if (t == ctl->t_start) {
04307
04308         /* Write info... */
04309         printf("Write station data: %s\n", filename);
04310
04311         /* Create new file... */
04312         if (!(out = fopen(filename, "w")))
04313             ERRMSG("Cannot create file!");
04314
04315         /* Write header... */
04316         fprintf(out,
04317             "# $1 = time [s]\n"
04318             "# $2 = altitude [km]\n"
04319             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
04320         for (int iq = 0; iq < ctl->nq; iq++)
04321             fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
04322                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
04323         fprintf(out, "\n");
04324
04325         /* Set geolocation and search radius... */
04326         geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
04327         rmax2 = SQR(ctl->stat_r);
04328     }
04329
04330     /* Set time interval for output... */
04331     t0 = t - 0.5 * ctl->dt_mod;
04332     t1 = t + 0.5 * ctl->dt_mod;
04333
04334     /* Loop over air parcels... */
04335     for (int ip = 0; ip < atm->np; ip++) {
04336
04337         /* Check time... */
04338         if (atm->time[ip] < t0 || atm->time[ip] > t1)
04339             continue;
04340
04341         /* Check station flag... */
04342         if (ctl->qnt_stat >= 0)
04343             if (atm->q[ctl->qnt_stat][ip])
04344                 continue;
04345
04346         /* Get Cartesian coordinates... */
04347         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
04348
04349         /* Check horizontal distance... */
04350         if (DIST2(x0, x1) > rmax2)
04351             continue;
04352
04353         /* Set station flag... */
04354         if (ctl->qnt_stat >= 0)
04355             atm->q[ctl->qnt_stat][ip] = 1;
  
```

```

04356
04357     /* Write data... */
04358     fprintf(out, "%.2f %g %g %g",
04359             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
04360     for (int iq = 0; iq < ctl->nq; iq++) {
04361         fprintf(out, " ");
04362         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
04363     }
04364     fprintf(out, "\n");
04365 }
04366
04367 /* Close file... */
04368 if (t == ctl->t_stop)
04369     fclose(out);
04370 }

```

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 /* -----
00036  Includes...
00037  ----- */
00038
00039 #include <ctype.h>
00040 #include <gsl/gsl_math.h>
00041 #include <gsl/gsl_randist.h>
00042 #include <gsl/gsl_rng.h>
00043 #include <gsl/gsl_sort.h>
00044 #include <gsl/gsl_spline.h>
00045 #include <gsl/gsl_statistics.h>
00046 #include <math.h>
00047 #include <netcdf.h>
00048 #include <omp.h>
00049 #include <stdio.h>
00050 #include <stdlib.h>
00051 #include <string.h>
00052 #include <time.h>
00053 #include <sys/time.h>
00054
00055 #ifdef MPI
00056 #include "mpi.h"
00057 #endif
00058
00059 #ifdef _OPENACC
00060 #include "openacc.h"

```



```

00061 #include "curand.h"
00062 #endif
00063
00064 /* -----
00065     Constants...
00066 ----- */
00067
00069 #define G0 9.80665
00070
00072 #define H0 7.0
00073
00075 #define KB 1.3806504e-23
00076
00078 #define MA 28.9644
00079
00081 #define MH2O 18.01528
00082
00084 #define MO3 48.00
00085
00087 #define P0 1013.25
00088
00090 #define T0 273.15
00091
00093 #define RA 287.058
00094
00096 #define RI 8.3144598
00097
00099 #define RE 6367.421
00100
00101 /* -----
00102     Dimensions...
00103 ----- */
00104
00106 #define LEN 5000
00107
00109 #define NP 10000000
00110
00112 #define NQ 12
00113
00115 #define EP 112
00116
00118 #define EX 1201
00119
00121 #define EY 601
00122
00124 #define GX 720
00125
00127 #define GY 360
00128
00130 #define GZ 100
00131
00133 #define NENS 2000
00134
00136 #define NTHREADS 512
00137
00138 /* -----
00139     Macros...
00140 ----- */
00141
00143 #define ALLOC(ptr, type, n) \
00144     if((ptr=calloc((size_t)(n), sizeof(type)))==NULL) \
00145         ERRMSG("Out of memory!");
00146
00148 #define DEG2DX(dlon, lat) \
00149     ((dlon) * M_PI * RE / 180. * cos((lat) / 180. * M_PI))
00150
00152 #define DEG2DY(dlat) \
00153     ((dlat) * M_PI * RE / 180.)
00154
00156 #define DP2DZ(dp, p) \
00157     (- (dp) * H0 / (p))
00158
00160 #define DX2DEG(dx, lat) \
00161     (((lat) < -89.999 || (lat) > 89.999) ? 0 \
00162      : (dx) * 180. / (M_PI * RE * cos((lat) / 180. * M_PI)))
00163
00165 #define DY2DEG(dy) \
00166     ((dy) * 180. / (M_PI * RE))
00167
00169 #define DZ2DP(dz, p) \
00170     (- (dz) * (p) / H0)
00171
00173 #define DIST(a, b) sqrt(DIST2(a, b))
00174
00176 #define DIST2(a, b) \
00177     ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))
00178

```

```

00180 #define DOTP(a, b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
00181
00183 #define ERRMSG(msg) {
00184     printf("\nError (%s, %s, %d): %s\n\n",
00185         __FILE__, __func__, __LINE__, msg);
00186     exit(EXIT_FAILURE);
00187 }
00188
00190 #define FMOD(x, y) \
00191     ((x) - (int) ((x) / (y)) * (y))
00192
00194 #define FREAD(ptr, type, size, out) {
00195     if(fread(ptr, sizeof(type), size, out)!=size)
00196         ERRMSG("Error while reading!");
00197 }
00198
00200 #define FWRITE(ptr, type, size, out) {
00201     if(fwrite(ptr, sizeof(type), size, out)!=size)
00202         ERRMSG("Error while writing!");
00203 }
00204
00206 #define LIN(x0, y0, x1, y1, x) \
00207     ((y0)+(y1)-(y0))/((x1)-(x0))*((x)-(x0))
00208
00210 #define NC(cmd) {
00211     if((cmd)!=NC_NOERR)
00212         ERRMSG(nc_strerror(cmd));
00213 }
00214
00216 #define NORM(a) sqrt(DOTP(a, a))
00217
00219 #define PRINT(format, var) \
00220     printf("Print (%s, %s, %d): %s= "format"\n",
00221         __FILE__, __func__, __LINE__, #var, var);
00222
00224 #define P(z) (P0 * exp(-(z) / H0))
00225
00227 #define RH(p, t, h2o) (0.263 * 100. * (p) * MH2O / MA * (h2o) \
00228     / exp(17.67 * ((t) - T0) / ((t) - 29.65)))
00229
00231 #define SQR(x) ((x)*(x))
00232
00234 #define THETA(p, t) ((t) * pow(1000. / (p), 0.286))
00235
00237 #define TOK(line, tok, format, var) {
00238     if((tok)=strtok((line), " \t")) {
00239         if(sscanf(tok, format, &(var))!=1) continue;
00240     } else ERRMSG("Error while reading!");
00241 }
00242
00244 #define TVIRT(t, h2o) ((t) * (1.0 + 0.609133 * (h2o) * MH2O / MA))
00245
00247 #define WARN(msg) {
00248     printf("\nWarning (%s, %s, %d): %s\n\n",
00249         __FILE__, __func__, __LINE__, msg);
00250 }
00251
00253 #define Z(p) (H0 * log(P0 / (p)))
00254
00255 /* -----
00256     Timers...
00257     ----- */
00258
00260 #define START_TIMER(id) timer(#id, id, 1)
00261
00263 #define STOP_TIMER(id) timer(#id, id, 2)
00264
00266 #define PRINT_TIMER(id) timer(#id, id, 3)
00267
00269 #define NTIMER 20
00270
00272 #define TIMER_INIT 1
00273
00275 #define TIMER_INPUT 2
00276
00278 #define TIMER_OUTPUT 3
00279
00281 #define TIMER_ADVECT 4
00282
00284 #define TIMER_DECAY 5
00285
00287 #define TIMER_DIFFMESO 6
00288
00290 #define TIMER_DIFFTURB 7
00291
00293 #define TIMER_ISOSURF 8
00294

```

```

00296 #define TIMER_METEO 9
00297
00299 #define TIMER_POSITION 10
00300
00302 #define TIMER_SEDI 11
00303
00305 #define TIMER_OHCHEM 12
00306
00308 #define TIMER_WETDEPO 13
00309
00311 #define TIMER_TOTAL 14
00312
00313 /* -----
00314     NVIDIA Tools Extension (NVTX)...
00315     ----- */
00316
00317 #ifndef USE_NVTX
00318 #include "nvToolsExt.h"
00319
00321 #define NVTX_CPU 0xFFADD8E6
00322
00324 #define NVTX_GPU 0xFF00008B
00325
00327 #define NVTX_H2D 0xFFFFFFFF00
00328
00330 #define NVTX_D2H 0xFFFF8800
00331
00333 #define NVTX_READ 0xFFFFCCCB
00334
00336 #define NVTX_WRITE 0xFF8B0000
00337
00339 #define NVTX_MISC 0xFF808080
00340
00342 #define RANGE_PUSH(range_title, range_color) { \
00343     nvtxEventAttributes_t eventAttrib = {0}; \
00344     eventAttrib.version = NVTX_VERSION; \
00345     eventAttrib.size = NVTX_EVENT_ATTRIB_STRUCT_SIZE; \
00346     eventAttrib.messageType = NVTX_MESSAGE_TYPE_ASCII; \
00347     eventAttrib.colorType = NVTX_COLOR_ARGB; \
00348     eventAttrib.color = range_color; \
00349     eventAttrib.message.ascii = range_title; \
00350     nvtxRangePushEx(&eventAttrib); \
00351 }
00352
00354 #define RANGE_POP { \
00355     nvtxRangePop(); \
00356 }
00357 #else
00358
00359 /* Empty definitions of RANGE_PUSH and RANGE_POP... */
00360 #define RANGE_PUSH(range_title, range_color) {}
00361 #define RANGE_POP {}
00362 #endif
00363
00364 /* -----
00365     Structs...
00366     ----- */
00367
00369 typedef struct {
00370
00372     int nq;
00373
00375     char qnt_name[NQ][LEN];
00376
00378     char qnt_unit[NQ][LEN];
00379
00381     char qnt_format[NQ][LEN];
00382
00384     int qnt_ens;
00385
00387     int qnt_m;
00388
00390     int qnt_rho;
00391
00393     int qnt_r;
00394
00396     int qnt_ps;
00397
00399     int qnt_pt;
00400
00402     int qnt_z;
00403
00405     int qnt_p;
00406
00408     int qnt_t;
00409
00411     int qnt_u;

```

```
00412
00414     int qnt_v;
00415
00417     int qnt_w;
00418
00420     int qnt_h2o;
00421
00423     int qnt_o3;
00424
00426     int qnt_lwc;
00427
00429     int qnt_iwc;
00430
00432     int qnt_pc;
00433
00435     int qnt_hno3;
00436
00438     int qnt_oh;
00439
00441     int qnt_rh;
00442
00444     int qnt_theta;
00445
00447     int qnt_vh;
00448
00450     int qnt_vz;
00451
00453     int qnt_pv;
00454
00456     int qnt_tice;
00457
00459     int qnt_tsts;
00460
00462     int qnt_tnat;
00463
00465     int qnt_stat;
00466
00468     int direction;
00469
00471     double t_start;
00472
00474     double t_stop;
00475
00477     double dt_mod;
00478
00480     double dt_met;
00481
00483     int met_dx;
00484
00486     int met_dy;
00487
00489     int met_dp;
00490
00492     int met_sx;
00493
00495     int met_sy;
00496
00498     int met_sp;
00499
00501     int met_np;
00502
00504     double met_p[EP];
00505
00508     int met_tropo;
00509
00511     char met_geopot[LEN];
00512
00514     double met_dt_out;
00515
00517     char met_stage[LEN];
00518
00521     int isosurf;
00522
00524     char balloon[LEN];
00525
00527     double turb_dx_trop;
00528
00530     double turb_dx_strat;
00531
00533     double turb_dz_trop;
00534
00536     double turb_dz_strat;
00537
00539     double turb_mesox;
00540
00542     double turb_mesoz;
00543
```

```
00545 char species[LEN];
00546
00548 double molmass;
00549
00551 double tdec_trop;
00552
00554 double tdec_strat;
00555
00557 double oh_chem[4];
00558
00560 double wet_depo[4];
00561
00563 double psc_h2o;
00564
00566 double psc_hno3;
00567
00569 char atm_basename[LEN];
00570
00572 char atm_gpfile[LEN];
00573
00575 double atm_dt_out;
00576
00578 int atm_filter;
00579
00581 int atm_stride;
00582
00584 int atm_type;
00585
00587 char csi_basename[LEN];
00588
00590 double csi_dt_out;
00591
00593 char csi_obsfile[LEN];
00594
00596 double csi_obsmin;
00597
00599 double csi_modmin;
00600
00602 int csi_nz;
00603
00605 double csi_z0;
00606
00608 double csi_z1;
00609
00611 int csi_nx;
00612
00614 double csi_lon0;
00615
00617 double csi_lon1;
00618
00620 int csi_ny;
00621
00623 double csi_lat0;
00624
00626 double csi_lat1;
00627
00629 char grid_basename[LEN];
00630
00632 char grid_gpfile[LEN];
00633
00635 double grid_dt_out;
00636
00638 int grid_sparse;
00639
00641 int grid_nz;
00642
00644 double grid_z0;
00645
00647 double grid_z1;
00648
00650 int grid_nx;
00651
00653 double grid_lon0;
00654
00656 double grid_lon1;
00657
00659 int grid_ny;
00660
00662 double grid_lat0;
00663
00665 double grid_lat1;
00666
00668 char prof_basename[LEN];
00669
00671 char prof_obsfile[LEN];
00672
00674 int prof_nz;
```

```
00675
00677 double prof_z0;
00678
00680 double prof_z1;
00681
00683 int prof_nx;
00684
00686 double prof_lon0;
00687
00689 double prof_lon1;
00690
00692 int prof_ny;
00693
00695 double prof_lat0;
00696
00698 double prof_lat1;
00699
00701 char ens_basename[LEN];
00702
00704 char stat_basename[LEN];
00705
00707 double stat_lon;
00708
00710 double stat_lat;
00711
00713 double stat_r;
00714
00715 } ctl_t;
00716
00718 typedef struct {
00719
00721 int np;
00722
00724 double time[NP];
00725
00727 double p[NP];
00728
00730 double lon[NP];
00731
00733 double lat[NP];
00734
00736 double q[NQ][NP];
00737
00738 } atm_t;
00739
00741 typedef struct {
00742
00744 float up[NP];
00745
00747 float vp[NP];
00748
00750 float wp[NP];
00751
00753 double iso_var[NP];
00754
00756 double iso_ps[NP];
00757
00759 double iso_ts[NP];
00760
00762 int iso_n;
00763
00765 double tsig[EX][EY][EP];
00766
00768 float usig[EX][EY][EP];
00769
00771 float vsig[EX][EY][EP];
00772
00774 float wsig[EX][EY][EP];
00775
00776 } cache_t;
00777
00779 typedef struct {
00780
00782 double time;
00783
00785 int nx;
00786
00788 int ny;
00789
00791 int np;
00792
00794 double lon[EX];
00795
00797 double lat[EY];
00798
00800 double p[EP];
00801
```

```

00803 float ps[EX][EY];
00804
00806 float zs[EX][EY];
00807
00809 float pt[EX][EY];
00810
00812 float pc[EX][EY];
00813
00815 float cl[EX][EY];
00816
00818 float z[EX][EY][EP];
00819
00821 float t[EX][EY][EP];
00822
00824 float u[EX][EY][EP];
00825
00827 float v[EX][EY][EP];
00828
00830 float w[EX][EY][EP];
00831
00833 float pv[EX][EY][EP];
00834
00836 float h2o[EX][EY][EP];
00837
00839 float o3[EX][EY][EP];
00840
00842 float lwc[EX][EY][EP];
00843
00845 float iwc[EX][EY][EP];
00846
00848 float pl[EX][EY][EP];
00849
00850 } met_t;
00851
00852 /* -----
00853    Functions...
00854    ----- */
00855
00857 void cart2geo(
00858     double *x,
00859     double *z,
00860     double *lon,
00861     double *lat);
00862
00864 #ifdef _OPENACC
00865 #pragma acc routine (check_finite)
00866 #endif
00867 int check_finite(
00868     const double x);
00869
00871 #ifdef _OPENACC
00872 #pragma acc routine (clim_hno3)
00873 #endif
00874 double clim_hno3(
00875     double t,
00876     double lat,
00877     double p);
00878
00880 #ifdef _OPENACC
00881 #pragma acc routine (clim_oh)
00882 #endif
00883 double clim_oh(
00884     double t,
00885     double lat,
00886     double p);
00887
00889 #ifdef _OPENACC
00890 #pragma acc routine (clim_tropo)
00891 #endif
00892 double clim_tropo(
00893     double t,
00894     double lat);
00895
00897 void day2doy(
00898     int year,
00899     int mon,
00900     int day,
00901     int *doy);
00902
00904 void doy2day(
00905     int year,
00906     int doy,
00907     int *mon,
00908     int *day);
00909
00911 void geo2cart(
00912     double z,

```

```
00913     double lon,
00914     double lat,
00915     double *x);
00916
00917 void get_met(
00918     ctl_t *ctl,
00919     char *metbase,
00920     double t,
00921     met_t **met0,
00922     met_t **met1);
00923
00924 void get_met_help(
00925     double t,
00926     int direct,
00927     char *metbase,
00928     double dt_met,
00929     char *filename);
00930
00931 void get_met_replace(
00932     char *orig,
00933     char *search,
00934     char *repl);
00935
00936 #ifdef _OPENACC
00937 #pragma acc routine (intpol_met_space_3d)
00938 #endif
00939 void intpol_met_space_3d(
00940     met_t *met,
00941     float array[EX][EY][EP],
00942     double p,
00943     double lon,
00944     double lat,
00945     double *var,
00946     int *ci,
00947     double *cw,
00948     int init);
00949
00950 #ifdef _OPENACC
00951 #pragma acc routine (intpol_met_space_2d)
00952 #endif
00953 void intpol_met_space_2d(
00954     met_t *met,
00955     float array[EX][EY],
00956     double lon,
00957     double lat,
00958     double *var,
00959     int *ci,
00960     double *cw,
00961     int init);
00962
00963 #ifdef _OPENACC
00964 #pragma acc routine (intpol_met_time_3d)
00965 #endif
00966 void intpol_met_time_3d(
00967     met_t *met0,
00968     float array0[EX][EY][EP],
00969     met_t *met1,
00970     float array1[EX][EY][EP],
00971     double ts,
00972     double p,
00973     double lon,
00974     double lat,
00975     double *var,
00976     int *ci,
00977     double *cw,
00978     int init);
00979
00980 #ifdef _OPENACC
00981 #pragma acc routine (intpol_met_time_2d)
00982 #endif
00983 void intpol_met_time_2d(
00984     met_t *met0,
00985     float array0[EX][EY],
00986     met_t *met1,
00987     float array1[EX][EY],
00988     double ts,
00989     double lon,
00990     double lat,
00991     double *var,
00992     int *ci,
00993     double *cw,
00994     int init);
00995
00996 void jsec2time(
00997     double jsec,
00998     int *year,
00999     int *mon,
```



```
01008     int *day,
01009     int *hour,
01010     int *min,
01011     int *sec,
01012     double *remain);
01013
01015 #ifdef _OPENACC
01016 #pragma acc routine (locate_irr)
01017 #endif
01018 int locate_irr(
01019     double **x,
01020     int n,
01021     double x);
01022
01024 #ifdef _OPENACC
01025 #pragma acc routine (locate_reg)
01026 #endif
01027 int locate_reg(
01028     double **x,
01029     int n,
01030     double x);
01031
01033 int read_atm(
01034     const char *filename,
01035     ctl_t * ctl,
01036     atm_t * atm);
01037
01039 void read_ctl(
01040     const char *filename,
01041     int argc,
01042     char *argv[],
01043     ctl_t * ctl);
01044
01046 int read_met(
01047     ctl_t * ctl,
01048     char *filename,
01049     met_t * met);
01050
01052 void read_met_cloud(
01053     met_t * met);
01054
01056 void read_met_extrapolate(
01057     met_t * met);
01058
01060 void read_met_geopot(
01061     met_t * met);
01062
01064 int read_met_help_3d(
01065     int ncid,
01066     char *varname,
01067     char *varname2,
01068     met_t * met,
01069     float dest[EX][EY][EP],
01070     float scl);
01071
01073 int read_met_help_2d(
01074     int ncid,
01075     char *varname,
01076     char *varname2,
01077     met_t * met,
01078     float dest[EX][EY],
01079     float scl);
01080
01082 void read_met_ml2pl(
01083     ctl_t * ctl,
01084     met_t * met,
01085     float var[EX][EY][EP]);
01086
01088 void read_met_periodic(
01089     met_t * met);
01090
01092 void read_met_pv(
01093     met_t * met);
01094
01096 void read_met_sample(
01097     ctl_t * ctl,
01098     met_t * met);
01099
01101 void read_met_surface(
01102     int ncid,
01103     met_t * met);
01104
01106 void read_met_tropo(
01107     ctl_t * ctl,
01108     met_t * met);
01109
01111 double scan_ctl(
```

```

01112     const char *filename,
01113     int argc,
01114     char *argv[],
01115     const char *varname,
01116     int arridx,
01117     const char *defvalue,
01118     char *value);
01119
01121 void spline(
01122     double *x,
01123     double *y,
01124     int n,
01125     double *x2,
01126     double *y2,
01127     int n2);
01128
01130 #ifdef _OPENACC
01131 #pragma acc routine (stddev)
01132 #endif
01133 double stddev(
01134     double *data,
01135     int n);
01136
01138 void time2jsec(
01139     int year,
01140     int mon,
01141     int day,
01142     int hour,
01143     int min,
01144     int sec,
01145     double remain,
01146     double *jsec);
01147
01149 void timer(
01150     const char *name,
01151     int id,
01152     int mode);
01153
01155 void write_atm(
01156     const char *filename,
01157     ctl_t * ctl,
01158     atm_t * atm,
01159     double t);
01160
01162 void write_csi(
01163     const char *filename,
01164     ctl_t * ctl,
01165     atm_t * atm,
01166     double t);
01167
01169 void write_ens(
01170     const char *filename,
01171     ctl_t * ctl,
01172     atm_t * atm,
01173     double t);
01174
01176 void write_grid(
01177     const char *filename,
01178     ctl_t * ctl,
01179     met_t * met0,
01180     met_t * met1,
01181     atm_t * atm,
01182     double t);
01183
01185 void write_prof(
01186     const char *filename,
01187     ctl_t * ctl,
01188     met_t * met0,
01189     met_t * met1,
01190     atm_t * atm,
01191     double t);
01192
01194 void write_station(
01195     const char *filename,
01196     ctl_t * ctl,
01197     atm_t * atm,
01198     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->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"

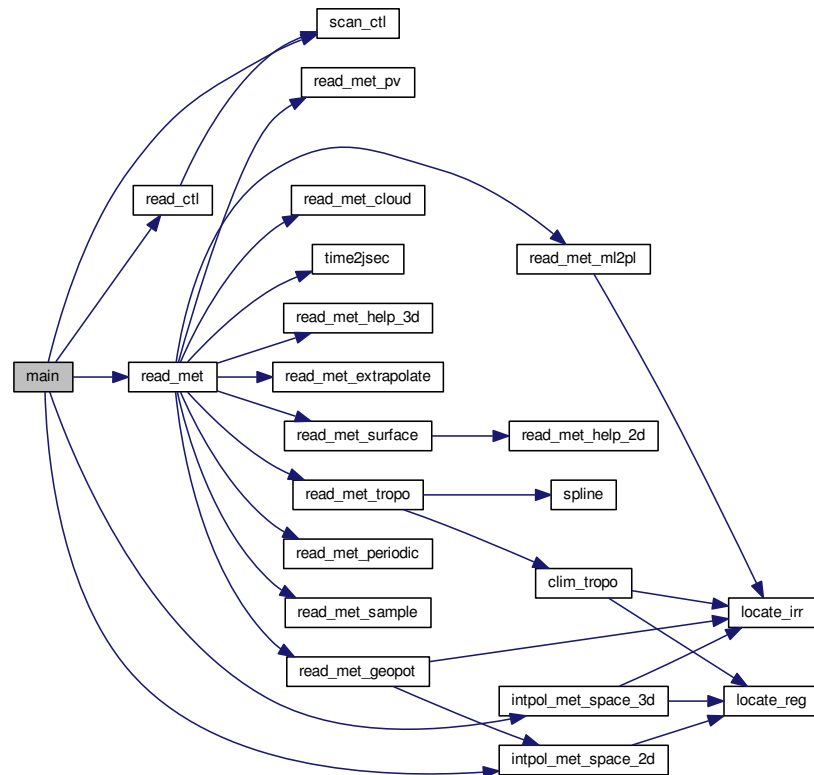
```

```

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\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
00025 #include "libtrac.h"
00026
00027 /* -----
00028  Dimensions...
00029  ----- */
00030
00032 #define NX 1441
00033
00035 #define NY 721
00036
00037 /* -----
00038  Main...
00039  ----- */
00040
00041 int main(

```

```

00042     int argc,
00043     char *argv[] {
00044
00045     ctl_t ctl;
00046
00047     met_t *met;
00048
00049     FILE *out;
00050
00051     static double timem[NX][NY], p0, ps, psm[NX][NY], 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;

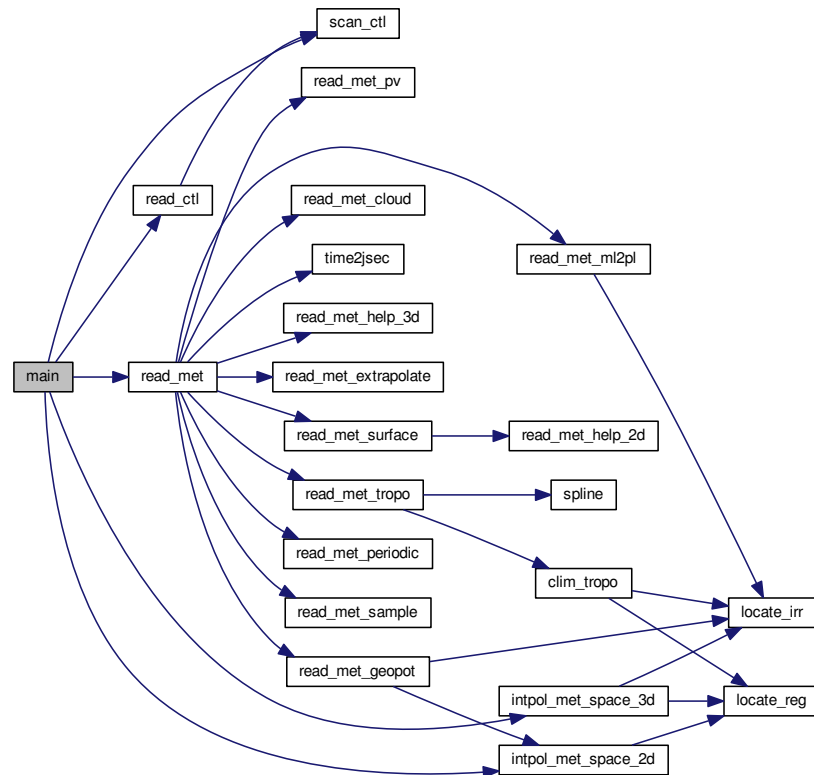
```

```

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\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\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);

```

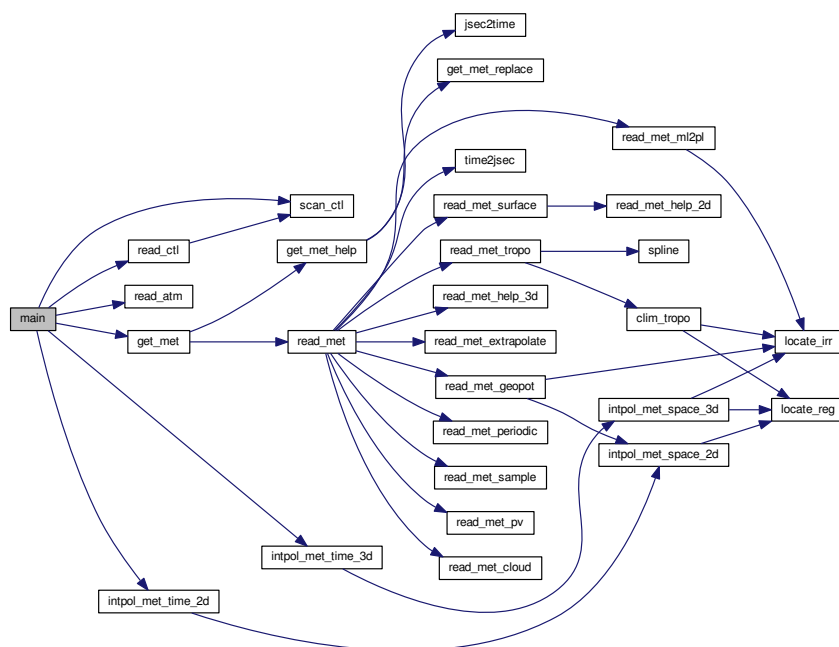


```

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\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->
time[ip], pref,
00113
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             }
00120             pref = 0.5 * (p0 + p1);
00121         }
00122
00123         /* Interpolate meteo data... */
00124         intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->
time[ip], pref,
00125
00126                                     atm->lon[ip], atm->lat[ip], &z, ci, cw, 1);
00127         intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip], pref,
00128
00129                                     atm->lon[ip], atm->lat[ip], &t, ci, cw, 0);
00130         intpol_met_time_3d(met0, met0->u, met1, met1->u, atm->
time[ip], pref,
00131
00132                                     atm->lon[ip], atm->lat[ip], &u, ci, cw, 0);
00133         intpol_met_time_3d(met0, met0->v, met1, met1->v, atm->
time[ip], pref,
00134
00135                                     atm->lon[ip], atm->lat[ip], &v, ci, cw, 0);
00136         intpol_met_time_3d(met0, met0->w, met1, met1->w, atm->
time[ip], pref,
00137
00138                                     atm->lon[ip], atm->lat[ip], &w, ci, cw, 0);
00139         intpol_met_time_3d(met0, met0->pv, met1, met1->pv, atm->
time[ip], pref,
00140
00141                                     atm->lon[ip], atm->lat[ip], &pv, ci, cw, 0);
00142         intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, atm->
time[ip], pref,
00143
00144                                     atm->lon[ip], atm->lat[ip], &h2o, ci, cw, 0);
00145         intpol_met_time_3d(met0, met0->o3, met1, met1->o3, atm->
time[ip], pref,
00146
00147                                     atm->lon[ip], atm->lat[ip], &o3, ci, cw, 0);
00148         intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, atm->
time[ip], pref,
00149
00150                                     atm->lon[ip], atm->lat[ip], &lwc, ci, cw, 0);
00151         intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, atm->
time[ip], pref,
00152
00153                                     atm->lon[ip], atm->lat[ip], &iwc, ci, cw, 0);
00154         intpol_met_time_2d(met0, met0->ps, met1, met1->ps, atm->
time[ip],
00155
00156                                     atm->lon[ip], atm->lat[ip], &ps, ci, cw, 0);
00157         intpol_met_time_2d(met0, met0->pt, met1, met1->pt, atm->
time[ip],
00158
00159                                     atm->lon[ip], atm->lat[ip], &pt, ci, cw, 0);
00160         intpol_met_time_2d(met0, met0->pc, met1, met1->pc, atm->
time[ip],
00161
00162                                     atm->lon[ip], atm->lat[ip], &pc, ci, cw, 0);
00163         intpol_met_time_2d(met0, met0->cl, met1, met1->cl, atm->
time[ip],
00164
00165                                     atm->lon[ip], atm->lat[ip], &cl, ci, cw, 0);
00166
00167         /* Interpolate tropopause data... */
00168         intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->
time[ip], pt,
00169
00170                                     atm->lon[ip], atm->lat[ip], &zt, ci, cw, 1);
00171         intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip], pt,
00172
00173                                     atm->lon[ip], atm->lat[ip], &tt, ci, cw, 0);
00174         intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, atm->
time[ip], pt,
00175
00176                                     atm->lon[ip], atm->lat[ip], &h2ot, ci, cw, 0);
00177
00178         /* Write data... */
00179         fprintf(out,
00180             "%.2f %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 }

```

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                     met->lat[iy], &z, ci, cw, 1);
00125                 lon[ix],
00126                     met->lat[iy], &t, ci, cw, 0);
00127                 lon[ix],
00128                     met->lat[iy], &u, ci, cw, 0);
00129                 lon[ix],
00130                     met->lat[iy], &v, ci, cw, 0);
00131                 lon[ix],
00132                     met->lat[iy], &w, ci, cw, 0);
00133                 lon[ix],
00134                     met->lat[iy], &pv, ci, cw, 0);
00135                 lon[ix],
00136                     met->lat[iy], &h2o, ci, cw, 0);
00137                 lon[ix],
00138                     met->lat[iy], &o3, ci, cw, 0);
00139                 lon[ix],
00140                     met->lat[iy], &lwc, ci, cw, 0);
00141                 lon[ix],
00142                     met->lat[iy], &iwc, ci, cw, 0);
00143                 lat[iy], &ps,
00144                     ci, cw, 0);
00145                 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->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\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],

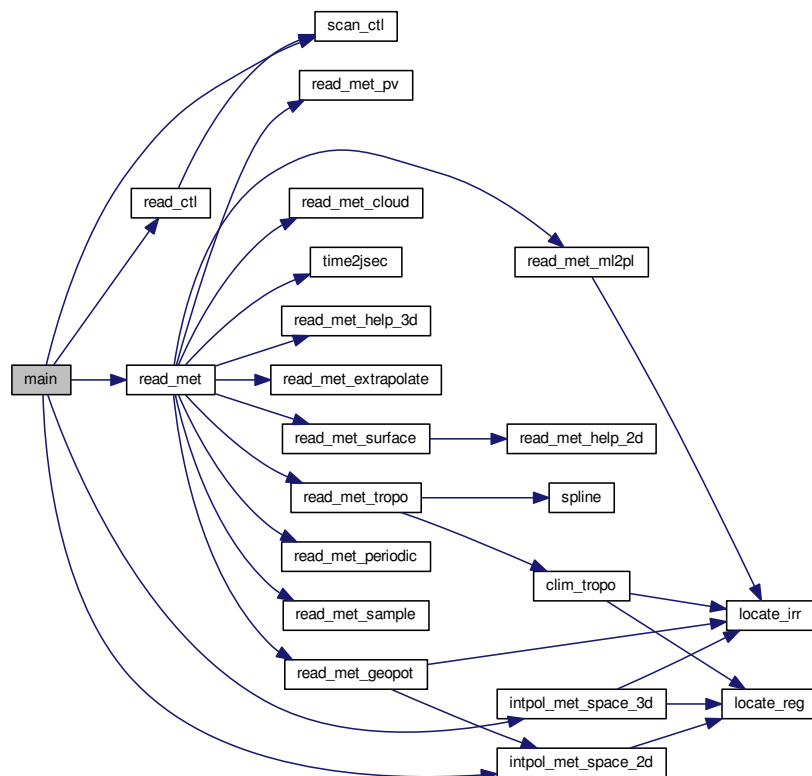
```

```

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], 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                 met->lat[iy], &z, ci, cw, 1);
00125             lon[ix],
00126                 met->lat[iy], &t, ci, cw, 0);
00127             lon[ix],
00128                 met->lat[iy], &u, ci, cw, 0);
00129             lon[ix],
00130                 met->lat[iy], &v, ci, cw, 0);
00131             lon[ix],
00132                 met->lat[iy], &w, ci, cw, 0);
00133             lon[ix],
00134                 met->lat[iy], &pv, ci, cw, 0);
00135             lon[ix],
00136                 met->lat[iy], &h2o, ci, cw, 0);
00137             lon[ix],
00138                 met->lat[iy], &o3, ci, cw, 0);
00139             lon[ix],
00140                 met->lat[iy], &lwc, ci, cw, 0);
00141             lon[ix],
00142                 met->lat[iy], &iwc, ci, cw, 0);
00143             lat[iy], &ps,
00144                 ci, cw, 0);
00145             lat[iy], &pt,
00146                 ci, cw, 0);
00147             lat[iy], &pc,
00148                 ci, cw, 0);
00149             lat[iy], &cl,
00150                 ci, cw, 0);
00151
00152             /* Interpolate tropopause data... */
00153             lat[iy],
00154                 &zt, ci, cw, 1);
00155             lat[iy],
00156                 &tt, ci, cw, 0);
00157             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 time2jsec.c File Reference

Convert date to Julian seconds.

Functions

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

5.31.1 Detailed Description

Convert date to Julian seconds.

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

5.31.2 Function Documentation

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

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

```

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

```

Here is the call graph for this function:



5.32 time2jsec.c

```

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

```

```

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 }

```

5.33 trac.c File Reference

Lagrangian particle dispersion model.

Functions

- void [module_advection](#) ([met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double *dt)
Calculate advection of air parcels.
- void [module_decay](#) ([ctl_t](#) *ctl, [atm_t](#) *atm, double *dt)
Calculate exponential decay of particle mass.
- void [module_diffusion_init](#) (void)
Initialize random number generator...
- void [module_diffusion_meso](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, [cache_t](#) *cache, double *dt, double *rs)
Calculate mesoscale diffusion.
- void [module_diffusion_rng](#) (double *rs, size_t n)
Generate random numbers.
- void [module_diffusion_turb](#) ([ctl_t](#) *ctl, [atm_t](#) *atm, double *dt, double *rs)
Calculate turbulent diffusion.
- void [module_isosurf_init](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, [cache_t](#) *cache)
Initialize isosurface module.
- void [module_isosurf](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, [cache_t](#) *cache)
Force air parcels to stay on isosurface.
- void [module_meteo](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm)
Interpolate meteorological data for air parcel positions.
- void [module_position](#) ([met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double *dt)
Check position of air parcels.
- void [module_sedi](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double *dt)
Calculate sedimentation of air parcels.
- void [module_oh_chem](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double *dt)
Calculate OH chemistry.
- void [module_wet_deposition](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double *dt)
Calculate wet deposition.
- void [write_output](#) (const char *dirname, [ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double t)
Write simulation output.
- int [main](#) (int argc, char *argv[])

Variables

- `curandGenerator_t rng`

5.33.1 Detailed Description

Lagrangian particle dispersion model.

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

5.33.2 Function Documentation

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

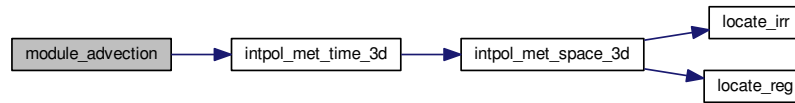
Calculate advection of air parcels.

Definition at line 507 of file [trac.c](#).

```

00511         {
00512
00513     #ifdef _OPENACC
00514     #pragma acc data present (met0,met1,atm,dt)
00515     #pragma acc parallel loop independent gang vector
00516     #else
00517     #pragma omp parallel for default(shared)
00518     #endif
00519     for (int ip = 0; ip < atm->np; ip++)
00520     if (dt[ip] != 0) {
00521
00522         int ci[3] = { 0 };
00523
00524         double dtm = 0.0, v[3] = { 0.0 }, xm[3] = {
00525             0.0 };
00526         double cw[3] = { 0.0 };
00527
00528         /* Interpolate meteorological data... */
00529         intpol_met_time_3d(met0, met0->u, met1, met1->u, atm->
time[ip],
00530             atm->p[ip], atm->lon[ip], atm->lat[ip], &v[0], ci,
00531             cw, 1);
00532         intpol_met_time_3d(met0, met0->v, met1, met1->v, atm->
time[ip],
00533             atm->p[ip], atm->lon[ip], atm->lat[ip], &v[1], ci,
00534             cw, 0);
00535         intpol_met_time_3d(met0, met0->w, met1, met1->w, atm->
time[ip],
00536             atm->p[ip], atm->lon[ip], atm->lat[ip], &v[2], ci,
00537             cw, 0);
00538
00539         /* Get position of the mid point... */
00540         dtm = atm->time[ip] + 0.5 * dt[ip];
00541         xm[0] =
00542             atm->lon[ip] + DX2DEG(0.5 * dt[ip] * v[0] / 1000., atm->lat[ip]);
00543         xm[1] = atm->lat[ip] + DY2DEG(0.5 * dt[ip] * v[1] / 1000.);
00544         xm[2] = atm->p[ip] + 0.5 * dt[ip] * v[2];
00545
00546         /* Interpolate meteorological data for mid point... */
00547         intpol_met_time_3d(met0, met0->u, met1, met1->u, dtm, xm[2], xm[0],
00548             xm[1], &v[0], ci, cw, 1);
00549         intpol_met_time_3d(met0, met0->v, met1, met1->v, dtm, xm[2], xm[0],
00550             xm[1], &v[1], ci, cw, 0);
00551         intpol_met_time_3d(met0, met0->w, met1, met1->w, dtm, xm[2], xm[0],
00552             xm[1], &v[2], ci, cw, 0);
00553
00554         /* Save new position... */
00555         atm->time[ip] += dt[ip];
00556         atm->lon[ip] += DX2DEG(dt[ip] * v[0] / 1000., xm[1]);
00557         atm->lat[ip] += DY2DEG(dt[ip] * v[1] / 1000.);
00558         atm->p[ip] += dt[ip] * v[2];
00559     }
00560 }
```

Here is the call graph for this function:



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

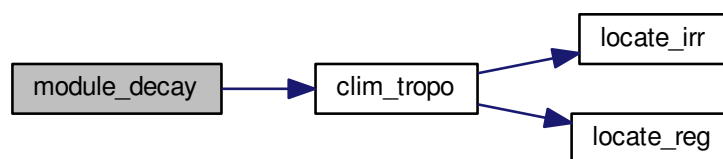
Calculate exponential decay of particle mass.

Definition at line 564 of file [trac.c](#).

```

00567     {
00568
00569     /* Check quantity flags... */
00570     if (ctl->qnt_m < 0)
00571         ERRMSG("Module needs quantity mass!");
00572
00573 #ifdef _OPENACC
00574 #pragma acc data present(ctl,atm,dt)
00575 #pragma acc parallel loop independent gang vector
00576 #else
00577 #pragma omp parallel for default(shared)
00578 #endif
00579     for (int ip = 0; ip < atm->np; ip++)
00580     if (dt[ip] != 0) {
00581
00582         double p0, p1, pt, tdec, w;
00583
00584         /* Get tropopause pressure... */
00585         pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00586
00587         /* Get weighting factor... */
00588         p1 = pt * 0.866877899;
00589         p0 = pt / 0.866877899;
00590         if (atm->p[ip] > p0)
00591             w = 1;
00592         else if (atm->p[ip] < p1)
00593             w = 0;
00594         else
00595             w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00596
00597         /* Set lifetime... */
00598         tdec = w * ctl->tdec_tropo + (1 - w) * ctl->tdec_strat;
00599
00600         /* Calculate exponential decay... */
00601         atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] / tdec);
00602     }
00603 }
  
```

Here is the call graph for this function:



5.33.2.3 void module_diffusion_init (void)

Initialize random number generator...

Definition at line 607 of file [trac.c](#).

```

00608     {
00609
00610     /* Initialize random number generator... */
00611     #ifdef _OPENACC
00612
00613     if (curandCreateGenerator(&rng, CURAND_RNG_PSEUDO_DEFAULT)
00614         != CURAND_STATUS_SUCCESS)
00615         ERRMSG("Cannot create random number generator!");
00616     if (curandSetStream(rng, (cudaStream_t) acc_get_cuda_stream(acc_async_sync))
00617         != CURAND_STATUS_SUCCESS)
00618         ERRMSG("Cannot set stream for random number generator!");
00619
00620     #else
00621
00622     gsl_rng_env_setup();
00623     if (omp_get_max_threads() > NTHREADS)
00624         ERRMSG("Too many threads!");
00625     for (int i = 0; i < NTHREADS; i++) {
00626         rng[i] = gsl_rng_alloc(gsl_rng_default);
00627         gsl_rng_set(rng[i], gsl_rng_default_seed + (long unsigned) i);
00628     }
00629
00630     #endif
00631 }

```

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

Calculate mesoscale diffusion.

Definition at line 635 of file [trac.c](#).

```

00642     {
00643
00644     #ifdef _OPENACC
00645     #pragma acc data present(ctl,met0,met1,atm,cache,dt,rs)
00646     #pragma acc parallel loop independent gang vector
00647     #else
00648     #pragma omp parallel for default(shared)
00649     #endif
00650     for (int ip = 0; ip < atm->np; ip++)
00651         if (dt[ip] != 0) {
00652
00653             double u[16], v[16], w[16];
00654
00655             /* Get indices... */
00656             int ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
00657             int iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
00658             int iz = locate_irr(met0->p, met0->np, atm->p[ip]);
00659
00660             /* Caching of wind standard deviations... */
00661             if (cache->tsig[ix][iy][iz] != met0->time) {
00662
00663                 /* Collect local wind data... */
00664                 u[0] = met0->u[ix][iy][iz];
00665                 u[1] = met0->u[ix + 1][iy][iz];
00666                 u[2] = met0->u[ix][iy + 1][iz];
00667                 u[3] = met0->u[ix + 1][iy + 1][iz];
00668                 u[4] = met0->u[ix][iy][iz + 1];
00669                 u[5] = met0->u[ix + 1][iy][iz + 1];
00670                 u[6] = met0->u[ix][iy + 1][iz + 1];
00671                 u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00672
00673                 v[0] = met0->v[ix][iy][iz];
00674                 v[1] = met0->v[ix + 1][iy][iz];
00675                 v[2] = met0->v[ix][iy + 1][iz];
00676                 v[3] = met0->v[ix + 1][iy + 1][iz];
00677                 v[4] = met0->v[ix][iy][iz + 1];
00678                 v[5] = met0->v[ix + 1][iy][iz + 1];
00679                 v[6] = met0->v[ix][iy + 1][iz + 1];

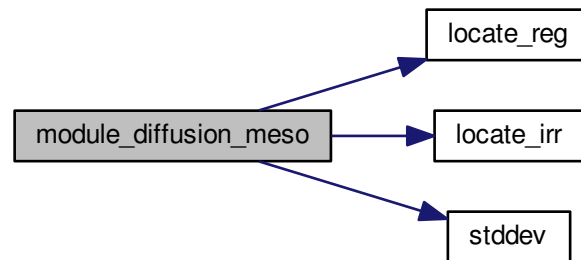
```

```

00680     v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00681
00682     w[0] = met0->w[ix][iy][iz];
00683     w[1] = met0->w[ix + 1][iy][iz];
00684     w[2] = met0->w[ix][iy + 1][iz];
00685     w[3] = met0->w[ix + 1][iy + 1][iz];
00686     w[4] = met0->w[ix][iy][iz + 1];
00687     w[5] = met0->w[ix + 1][iy][iz + 1];
00688     w[6] = met0->w[ix][iy + 1][iz + 1];
00689     w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00690
00691     /* Collect local wind data... */
00692     u[8] = met1->u[ix][iy][iz];
00693     u[9] = met1->u[ix + 1][iy][iz];
00694     u[10] = met1->u[ix][iy + 1][iz];
00695     u[11] = met1->u[ix + 1][iy + 1][iz];
00696     u[12] = met1->u[ix][iy][iz + 1];
00697     u[13] = met1->u[ix + 1][iy][iz + 1];
00698     u[14] = met1->u[ix][iy + 1][iz + 1];
00699     u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00700
00701     v[8] = met1->v[ix][iy][iz];
00702     v[9] = met1->v[ix + 1][iy][iz];
00703     v[10] = met1->v[ix][iy + 1][iz];
00704     v[11] = met1->v[ix + 1][iy + 1][iz];
00705     v[12] = met1->v[ix][iy][iz + 1];
00706     v[13] = met1->v[ix + 1][iy][iz + 1];
00707     v[14] = met1->v[ix][iy + 1][iz + 1];
00708     v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00709
00710     w[8] = met1->w[ix][iy][iz];
00711     w[9] = met1->w[ix + 1][iy][iz];
00712     w[10] = met1->w[ix][iy + 1][iz];
00713     w[11] = met1->w[ix + 1][iy + 1][iz];
00714     w[12] = met1->w[ix][iy][iz + 1];
00715     w[13] = met1->w[ix + 1][iy][iz + 1];
00716     w[14] = met1->w[ix][iy + 1][iz + 1];
00717     w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00718
00719     /* Get standard deviations of local wind data... */
00720     cache->usig[ix][iy][iz] = (float) stddev(u, 16);
00721     cache->vsig[ix][iy][iz] = (float) stddev(v, 16);
00722     cache->wsig[ix][iy][iz] = (float) stddev(w, 16);
00723     cache->tsig[ix][iy][iz] = met0->time;
00724 }
00725
00726 /* Set temporal correlations for mesoscale fluctuations... */
00727 double r = 1 - 2 * fabs(dt[ip]) / ctl->dt_met;
00728 double r2 = sqrt(1 - r * r);
00729
00730 /* Calculate horizontal mesoscale wind fluctuations... */
00731 if (ctl->turb_mesox > 0) {
00732     cache->up[ip] = (float)
00733         (r * cache->up[ip]
00734          + r2 * rs[3 * ip] * ctl->turb_mesox * cache->usig[ix][iy][iz]);
00735     atm->lon[ip] += DX2DEG(cache->up[ip] * dt[ip] / 1000., atm->lat[ip]);
00736
00737     cache->vp[ip] = (float)
00738         (r * cache->vp[ip]
00739          + r2 * rs[3 * ip + 1] * ctl->turb_mesox * cache->vsig[ix][iy][iz]);
00740     atm->lat[ip] += DY2DEG(cache->vp[ip] * dt[ip] / 1000.);
00741 }
00742
00743 /* Calculate vertical mesoscale wind fluctuations... */
00744 if (ctl->turb_mesoz > 0) {
00745     cache->wp[ip] = (float)
00746         (r * cache->wp[ip]
00747          + r2 * rs[3 * ip + 2] * ctl->turb_mesoz * cache->wsig[ix][iy][iz]);
00748     atm->p[ip] += cache->wp[ip] * dt[ip];
00749 }
00750 }
00751 }

```


Here is the call graph for this function:



5.33.2.5 void module_diffusion_rng (double * rs, size_t n)

Generate random numbers.

Definition at line 755 of file [trac.c](#).

```

00757     {
00758
00759     #ifdef _OPENACC
00760
00761     #pragma acc host_data use_device(rs)
00762     {
00763         if (curandGenerateNormalDouble(rng, rs, n, 0.0, 1.0)
00764             != CURAND_STATUS_SUCCESS)
00765             ERRMSG("Cannot create random numbers!");
00766     }
00767
00768     #else
00769
00770     #pragma omp parallel for default(shared)
00771     for (size_t i = 0; i < n; ++i)
00772         rs[i] = gsl_ran_gaussian_ziggurat(rng[omp_get_thread_num()], 1.0);
00773
00774     #endif
00775
00776 }
  
```

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

Calculate turbulent diffusion.

Definition at line 780 of file [trac.c](#).

```

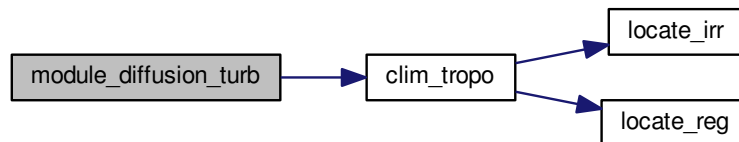
00784     {
00785
00786     #ifdef _OPENACC
00787     #pragma acc data present(ctl,atm,dt,rs)
00788     #pragma acc parallel loop independent gang vector
00789     #else
00790     #pragma omp parallel for default(shared)
00791     #endif
00792     for (int ip = 0; ip < atm->np; ip++)
00793         if (dt[ip] != 0) {
00794
00795             double w;
00796
  
```

```

00797      /* Get tropopause pressure... */
00798      double pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00799
00800      /* Get weighting factor... */
00801      double p1 = pt * 0.866877899;
00802      double p0 = pt / 0.866877899;
00803      if (atm->p[ip] > p0)
00804          w = 1;
00805      else if (atm->p[ip] < p1)
00806          w = 0;
00807      else
00808          w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00809
00810      /* Set diffusivity... */
00811      double dx = w * ctl->turb_dx_trop + (1 - w) * ctl->
turb_dx_strat;
00812      double dz = w * ctl->turb_dz_trop + (1 - w) * ctl->
turb_dz_strat;
00813
00814      /* Horizontal turbulent diffusion... */
00815      if (dx > 0) {
00816          double sigma = sqrt(2.0 * dx * fabs(dt[ip]));
00817          atm->lon[ip] += DX2DEG(rs[3 * ip] * sigma / 1000., atm->lat[ip]);
00818          atm->lat[ip] += DY2DEG(rs[3 * ip + 1] * sigma / 1000.);
00819      }
00820
00821      /* Vertical turbulent diffusion... */
00822      if (dz > 0) {
00823          double sigma = sqrt(2.0 * dz * fabs(dt[ip]));
00824          atm->p[ip]
00825              += DZ2DP(rs[3 * ip + 2] * sigma / 1000., atm->p[ip]);
00826      }
00827  }
00828 }

```

Here is the call graph for this function:



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

Initialize isosurface module.

Definition at line 832 of file [trac.c](#).

```

00837      {
00838
00839      FILE *in;
00840
00841      char line[LEN];
00842
00843      double t, cw[3];
00844
00845      int ci[3];
00846
00847      /* Save pressure... */
00848      if (ctl->isosurf == 1)
00849          for (int ip = 0; ip < atm->np; ip++)
00850              cache->iso_var[ip] = atm->p[ip];
00851
00852      /* Save density... */

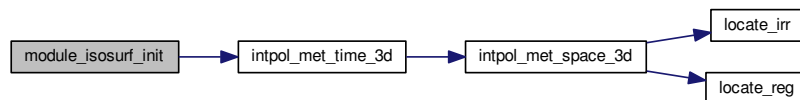
```

```

00853     else if (ctl->isosurf == 2)
00854         for (int ip = 0; ip < atm->np; ip++) {
00855             intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
00856                             atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
00857                             1);
00858             cache->iso_var[ip] = atm->p[ip] / t;
00859         }
00860
00861     /* Save potential temperature... */
00862     else if (ctl->isosurf == 3)
00863         for (int ip = 0; ip < atm->np; ip++) {
00864             intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
00865                             atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
00866                             1);
00867             cache->iso_var[ip] = THETA(atm->p[ip], t);
00868         }
00869
00870     /* Read balloon pressure data... */
00871     else if (ctl->isosurf == 4) {
00872
00873         /* Write info... */
00874         printf("Read balloon pressure data: %s\n", ctl->balloon);
00875
00876         /* Open file... */
00877         if (!(in = fopen(ctl->balloon, "r")))
00878             ERRMSG("Cannot open file!");
00879
00880         /* Read pressure time series... */
00881         while (fgets(line, LEN, in))
00882             if (sscanf(line, "%lg %lg", &(cache->iso_ts[cache->iso_n]),
00883                     &(cache->iso_ps[cache->iso_n])) == 2)
00884                 if (++cache->iso_n > NP)
00885                     ERRMSG("Too many data points!");
00886
00887         /* Check number of points... */
00888         if (cache->iso_n < 1)
00889             ERRMSG("Could not read any data!");
00890
00891         /* Close file... */
00892         fclose(in);
00893     }
00894 }

```

Here is the call graph for this function:



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

Force air parcels to stay on isosurface.

Definition at line 898 of file [trac.c](#).

```

00903     {
00904
00905     #ifdef _OPENACC
00906     #pragma acc data present(ctl,met0,met1,atm,cache)
00907     #pragma acc parallel loop independent gang vector
00908     #else
00909     #pragma omp parallel for default(shared)
00910     #endif
00911     for (int ip = 0; ip < atm->np; ip++) {
00912
00913         double t, cw[3];

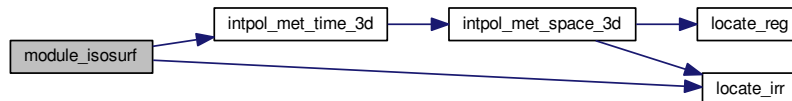
```

```

00914
00915     int ci[3];
00916
00917     /* Restore pressure... */
00918     if (ctl->isosurf == 1)
00919         atm->p[ip] = cache->iso_var[ip];
00920
00921     /* Restore density... */
00922     else if (ctl->isosurf == 2) {
00923         intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
00924                             atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
00925                             1);
00926         atm->p[ip] = cache->iso_var[ip] * t;
00927     }
00928
00929     /* Restore potential temperature... */
00930     else if (ctl->isosurf == 3) {
00931         intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
00932                             atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
00933                             1);
00934         atm->p[ip] = 1000. * pow(cache->iso_var[ip] / t, -1. / 0.286);
00935     }
00936
00937     /* Interpolate pressure... */
00938     else if (ctl->isosurf == 4) {
00939         if (atm->time[ip] <= cache->iso_ts[0])
00940             atm->p[ip] = cache->iso_ps[0];
00941         else if (atm->time[ip] >= cache->iso_ts[cache->iso_n - 1])
00942             atm->p[ip] = cache->iso_ps[cache->iso_n - 1];
00943         else {
00944             int idx = locate_irr(cache->iso_ts, cache->iso_n, atm->
time[ip]);
00945             atm->p[ip] = LIN(cache->iso_ts[idx], cache->iso_ps[idx],
00946                             cache->iso_ts[idx + 1], cache->iso_ps[idx + 1],
00947                             atm->time[ip]);
00948         }
00949     }
00950 }
00951 }

```

Here is the call graph for this function:



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

Interpolate meteorological data for air parcel positions.

Definition at line 955 of file [trac.c](#).

```

00959     {
00960
00961         /* Check quantity flags... */
00962         if (ctl->qnt_tsts >= 0)
00963             if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
00964                 ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
00965
00966         #ifdef _OPENACC
00967         #pragma acc data present(ctl,met0,met1,atm)
00968         #pragma acc parallel loop independent gang vector
00969         #else
00970         #pragma omp parallel for default(shared)
00971         #endif
00972         for (int ip = 0; ip < atm->np; ip++) {

```

```

00973
00974     double ps, pt, pc, pv, t, u, v, w, h2o, o3, lwc, iwc, z, cw[3];
00975
00976     int ci[3];
00977
00978     /* Interpolate meteorological data... */
00979     intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->
time[ip],
00980                       atm->p[ip], atm->lon[ip], atm->lat[ip], &z, ci, cw, 1);
00981     intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
00982                       atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw, 0);
00983     intpol_met_time_3d(met0, met0->u, met1, met1->u, atm->
time[ip],
00984                       atm->p[ip], atm->lon[ip], atm->lat[ip], &u, ci, cw, 0);
00985     intpol_met_time_3d(met0, met0->v, met1, met1->v, atm->
time[ip],
00986                       atm->p[ip], atm->lon[ip], atm->lat[ip], &v, ci, cw, 0);
00987     intpol_met_time_3d(met0, met0->w, met1, met1->w, atm->
time[ip],
00988                       atm->p[ip], atm->lon[ip], atm->lat[ip], &w, ci, cw, 0);
00989     intpol_met_time_3d(met0, met0->pv, met1, met1->pv, atm->
time[ip],
00990                       atm->p[ip], atm->lon[ip], atm->lat[ip], &pv, ci, cw,
00991                       0);
00992     intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, atm->
time[ip],
00993                       atm->p[ip], atm->lon[ip], atm->lat[ip], &h2o, ci, cw,
00994                       0);
00995     intpol_met_time_3d(met0, met0->o3, met1, met1->o3, atm->
time[ip],
00996                       atm->p[ip], atm->lon[ip], atm->lat[ip], &o3, ci, cw,
00997                       0);
00998     intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, atm->
time[ip],
00999                       atm->p[ip], atm->lon[ip], atm->lat[ip], &lwc, ci, cw,
01000                       0);
01001     intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, atm->
time[ip],
01002                       atm->p[ip], atm->lon[ip], atm->lat[ip], &iwc, ci, cw,
01003                       0);
01004     intpol_met_time_2d(met0, met0->ps, met1, met1->ps, atm->
time[ip],
01005                       atm->lon[ip], atm->lat[ip], &ps, ci, cw, 0);
01006     intpol_met_time_2d(met0, met0->pt, met1, met1->pt, atm->
time[ip],
01007                       atm->lon[ip], atm->lat[ip], &pt, ci, cw, 0);
01008     intpol_met_time_2d(met0, met0->pc, met1, met1->pc, atm->
time[ip],
01009                       atm->lon[ip], atm->lat[ip], &pc, ci, cw, 0);
01010
01011     /* Set surface pressure... */
01012     if (ctl->qnt_ps >= 0)
01013         atm->q[ctl->qnt_ps][ip] = ps;
01014
01015     /* Set tropopause pressure... */
01016     if (ctl->qnt_pt >= 0)
01017         atm->q[ctl->qnt_pt][ip] = pt;
01018
01019     /* Set pressure... */
01020     if (ctl->qnt_p >= 0)
01021         atm->q[ctl->qnt_p][ip] = atm->p[ip];
01022
01023     /* Set geopotential height... */
01024     if (ctl->qnt_z >= 0)
01025         atm->q[ctl->qnt_z][ip] = z;
01026
01027     /* Set temperature... */
01028     if (ctl->qnt_t >= 0)
01029         atm->q[ctl->qnt_t][ip] = t;
01030
01031     /* Set zonal wind... */
01032     if (ctl->qnt_u >= 0)
01033         atm->q[ctl->qnt_u][ip] = u;
01034
01035     /* Set meridional wind... */
01036     if (ctl->qnt_v >= 0)
01037         atm->q[ctl->qnt_v][ip] = v;
01038
01039     /* Set vertical velocity... */
01040     if (ctl->qnt_w >= 0)
01041         atm->q[ctl->qnt_w][ip] = w;
01042
01043     /* Set water vapor vmr... */
01044     if (ctl->qnt_h2o >= 0)
01045         atm->q[ctl->qnt_h2o][ip] = h2o;
01046

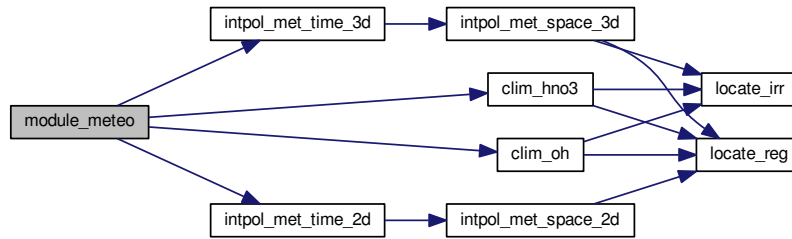
```

```

01047      /* Set ozone vmr... */
01048      if (ctl->qnt_o3 >= 0)
01049          atm->q[ctl->qnt_o3][ip] = o3;
01050
01051      /* Set cloud liquid water content... */
01052      if (ctl->qnt_lwc >= 0)
01053          atm->q[ctl->qnt_lwc][ip] = lwc;
01054
01055      /* Set cloud ice water content... */
01056      if (ctl->qnt_iwc >= 0)
01057          atm->q[ctl->qnt_iwc][ip] = iwc;
01058
01059      /* Set cloud top pressure... */
01060      if (ctl->qnt_pc >= 0)
01061          atm->q[ctl->qnt_pc][ip] = pc;
01062
01063      /* Set nitric acid vmr... */
01064      if (ctl->qnt_hno3 >= 0)
01065          atm->q[ctl->qnt_hno3][ip] =
01066              clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip]);
01067
01068      /* Set hydroxyl number concentration... */
01069      if (ctl->qnt_oh >= 0)
01070          atm->q[ctl->qnt_oh][ip] =
01071              clim_oh(atm->time[ip], atm->lat[ip], atm->p[ip]);
01072
01073      /* Calculate horizontal wind... */
01074      if (ctl->qnt_vh >= 0)
01075          atm->q[ctl->qnt_vh][ip] = sqrt(u * u + v * v);
01076
01077      /* Calculate vertical velocity... */
01078      if (ctl->qnt_vz >= 0)
01079          atm->q[ctl->qnt_vz][ip] = -1e3 * H0 / atm->p[ip] * w;
01080
01081      /* Calculate relative humidity... */
01082      if (ctl->qnt_rh >= 0)
01083          atm->q[ctl->qnt_rh][ip] = RH(atm->p[ip], t, h2o);
01084
01085      /* Calculate potential temperature... */
01086      if (ctl->qnt_theta >= 0)
01087          atm->q[ctl->qnt_theta][ip] = THETA(atm->p[ip], t);
01088
01089      /* Set potential vorticity... */
01090      if (ctl->qnt_pv >= 0)
01091          atm->q[ctl->qnt_pv][ip] = pv;
01092
01093      /* Calculate T_ice (Marti and Mauersberger, 1993)... */
01094      if (ctl->qnt_tice >= 0)
01095          atm->q[ctl->qnt_tice][ip] =
01096              -2663.5 /
01097              (log10((ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] * 100.) -
01098               12.537);
01099
01100      /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
01101      if (ctl->qnt_tnat >= 0) {
01102          double p_hno3;
01103          if (ctl->psc_hno3 > 0)
01104              p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
01105          else
01106              p_hno3 = clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip])
01107                  * 1e-9 * atm->p[ip] / 1.333224;
01108          double p_h2o =
01109              (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
01110          double a = 0.009179 - 0.00088 * log10(p_h2o);
01111          double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
01112          double c = -11397.0 / a;
01113          double x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
01114          double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
01115          if (x1 > 0)
01116              atm->q[ctl->qnt_tnat][ip] = x1;
01117          if (x2 > 0)
01118              atm->q[ctl->qnt_tnat][ip] = x2;
01119      }
01120
01121      /* Calculate T_STS (mean of T_ice and T_NAT)... */
01122      if (ctl->qnt_tsts >= 0)
01123          atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
01124                                          + atm->q[ctl->qnt_tnat][ip]);
01125      }
01126 }

```

Here is the call graph for this function:



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

Check position of air parcels.

Definition at line 1130 of file [trac.c](#).

```

01134         {
01135
01136 #ifdef _OPENACC
01137 #pragma acc data present (met0,met1,atm,dt)
01138 #pragma acc parallel loop independent gang vector
01139 #else
01140 #pragma omp parallel for default(shared)
01141 #endif
01142     for (int ip = 0; ip < atm->np; ip++)
01143     if (dt[ip] != 0) {
01144
01145         double ps, cw[3];
01146
01147         int ci[3];
01148
01149         /* Calculate modulo... */
01150         atm->lon[ip] = FMOD(atm->lon[ip], 360.);
01151         atm->lat[ip] = FMOD(atm->lat[ip], 360.);
01152
01153         /* Check latitude... */
01154         while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
01155             if (atm->lat[ip] > 90) {
01156                 atm->lat[ip] = 180 - atm->lat[ip];
01157                 atm->lon[ip] += 180;
01158             }
01159             if (atm->lat[ip] < -90) {
01160                 atm->lat[ip] = -180 - atm->lat[ip];
01161                 atm->lon[ip] += 180;
01162             }
01163         }
01164
01165         /* Check longitude... */
01166         while (atm->lon[ip] < -180)
01167             atm->lon[ip] += 360;
01168         while (atm->lon[ip] >= 180)
01169             atm->lon[ip] -= 360;
01170
01171         /* Check pressure... */
01172         if (atm->p[ip] < met0->p[met0->np - 1])
01173             atm->p[ip] = met0->p[met0->np - 1];
01174         else if (atm->p[ip] > 300.) {
01175             intpol_met_time_2d(met0, met0->ps, met1, met1->ps, atm->
time[ip],
01176                             atm->lon[ip], atm->lat[ip], &ps, ci, cw, 1);
01177             if (atm->p[ip] > ps)
01178                 atm->p[ip] = ps;
01179         }
01180     }
01181 }

```

Here is the call graph for this function:



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

Calculate sedimentation of air parcels.

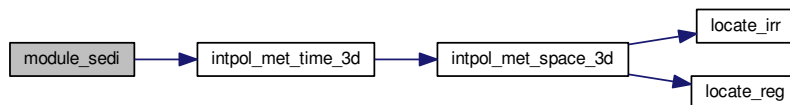
Definition at line 1185 of file [trac.c](#).

```

01190         {
01191
01192 #ifdef _OPENACC
01193 #pragma acc data present(ctl,met0,met1,atm,dt)
01194 #pragma acc parallel loop independent gang vector
01195 #else
01196 #pragma omp parallel for default(shared)
01197 #endif
01198     for (int ip = 0; ip < atm->np; ip++)
01199     if (dt[ip] != 0) {
01200
01201         double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p, cw[3];
01202
01203         int ci[3];
01204
01205         /* Convert units... */
01206         p = 100. * atm->p[ip];
01207         r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
01208         rho_p = atm->q[ctl->qnt_rho][ip];
01209
01210         /* Get temperature... */
01211         intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
01212                         atm->p[ip], atm->lon[ip], atm->lat[ip], &T, ci, cw,
01213                         1);
01214
01215         /* Density of dry air... */
01216         rho = p / (RA * T);
01217
01218         /* Dynamic viscosity of air... */
01219         eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
01220
01221         /* Thermal velocity of an air molecule... */
01222         v = sqrt(8. * KB * T / (M_PI * 4.8096e-26));
01223
01224         /* Mean free path of an air molecule... */
01225         lambda = 2. * eta / (rho * v);
01226
01227         /* Knudsen number for air... */
01228         K = lambda / r_p;
01229
01230         /* Cunningham slip-flow correction... */
01231         G = 1. + K * (1.249 + 0.42 * exp(-0.87 / K));
01232
01233         /* Sedimentation (fall) velocity... */
01234         v_p = 2. * SQR(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;
01235
01236         /* Calculate pressure change... */
01237         atm->p[ip] += DZ2DP(v_p * dt[ip] / 1000., atm->p[ip]);
01238     }
01239 }

```


Here is the call graph for this function:



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

Calculate OH chemistry.

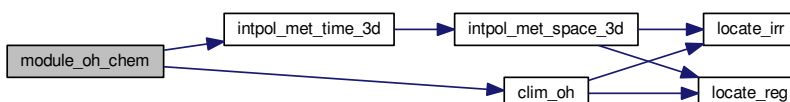
Definition at line 1243 of file trac.c.

```

01248     {
01249
01250     /* Check quantity flags... */
01251     if (ctl->qnt_m < 0)
01252         ERRMSG("Module needs quantity mass!");
01253
01254     #ifdef _OPENACC
01255     #pragma acc data present(ctl,met0,met1,atm,dt)
01256     #pragma acc parallel loop independent gang vector
01257     #else
01258     #pragma omp parallel for default(shared)
01259     #endif
01260     for (int ip = 0; ip < atm->np; ip++)
01261         if (dt[ip] != 0) {
01262
01263             double c, k, k0, ki, M, T, cw[3];
01264
01265             int ci[3];
01266
01267             /* Get temperature... */
01268             intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
01269                             atm->p[ip], atm->lon[ip], atm->lat[ip], &T, ci, cw,
1);
01270
01271
01272             /* Calculate molecular density... */
01273             M = 7.243e21 * (atm->p[ip] / P0) / T;
01274
01275             /* Calculate rate coefficient for X + OH + M -> XOH + M
(JPL Publication 15-10) ... */
01276             k0 = ctl->oh_chem[0] *
(ctl->oh_chem[1] > 0 ? pow(T / 300., -ctl->oh_chem[1]) : 1.);
01277             ki = ctl->oh_chem[2] *
(ctl->oh_chem[3] > 0 ? pow(T / 300., -ctl->oh_chem[3]) : 1.);
01278             c = log10(k0 * M / ki);
01279             k = k0 * M / (1. + k0 * M / ki) * pow(0.6, 1. / (1. + c * c));
01280
01281             /* Calculate exponential decay... */
01282             atm->q[ctl->qnt_m][ip] *=
exp(-dt[ip] * k * clim_oh(atm->time[ip], atm->lat[ip], atm->
p[ip]));
01283
01284         }
01285     }
01286 }
01287
01288 }

```

Here is the call graph for this function:



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

Calculate wet deposition.

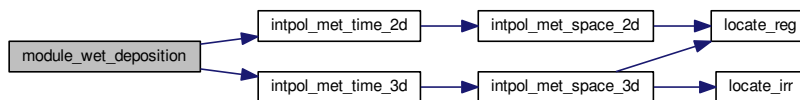
Definition at line 1292 of file [trac.c](#).

```

01297         {
01298
01299     /* Check quantity flags... */
01300     if (ctl->qnt_m < 0)
01301         ERRMSG("Module needs quantity mass!");
01302
01303 #ifdef _OPENACC
01304 #pragma acc data present(ctl,met0,met1,atm,dt)
01305 #pragma acc parallel loop independent gang vector
01306 #else
01307 #pragma omp parallel for default(shared)
01308 #endif
01309     for (int ip = 0; ip < atm->np; ip++)
01310         if (dt[ip] != 0) {
01311
01312             double H, Is, Si, T, cl, lambda, iwc, lwc, pc, cw[3];
01313
01314             int inside, ci[3];
01315
01316             /* Check whether particle is below cloud top... */
01317             intpol_met_time_2d(met0, met0->pc, met1, met1->pc, atm->
time[ip],
01318                             atm->lon[ip], atm->lat[ip], &pc, ci, cw, 1);
01319             if (!isfinite(pc) || atm->p[ip] <= pc)
01320                 continue;
01321
01322             /* Check whether particle is inside or below cloud... */
01323             intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, atm->
time[ip],
01324                             atm->p[ip], atm->lon[ip], atm->lat[ip], &lwc, ci, cw,
1);
01325
01326             intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, atm->
time[ip],
01327                             atm->p[ip], atm->lon[ip], atm->lat[ip], &iwc, ci, cw,
0);
01328
01329             inside = (iwc > 0 || lwc > 0);
01330
01331             /* Estimate precipitation rate (Pisso et al., 2019)... */
01332             intpol_met_time_2d(met0, met0->cl, met1, met1->cl, atm->
time[ip],
01333                             atm->lon[ip], atm->lat[ip], &cl, ci, cw, 0);
01334             Is = pow(2. * cl, 1. / 0.36);
01335             if (Is < 0.01)
01336                 continue;
01337
01338             /* Calculate in-cloud scavenging for gases... */
01339             if (inside) {
01340
01341                 /* Get temperature... */
01342                 intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
01343                             atm->p[ip], atm->lon[ip], atm->lat[ip], &T, ci, cw,
0);
01344
01345                 /* Get Henry's constant (Sander, 2015)... */
01346                 H = ctl->wet_depo[2] * 101.325
01347                     * exp(ctl->wet_depo[3] * (1. / T - 1. / 298.15));
01348
01349                 /* Get scavenging coefficient (Hertel et al., 1995)... */
01350                 Si = 1. / ((1. - cl) / (H * RI / P0 * T) + cl);
01351                 lambda = 6.2 * Si * Is / 3.6e6;
01352             }
01353
01354             /* Calculate below-cloud scavenging for gases (Pisso et al., 2019)... */
01355             else
01356                 lambda = ctl->wet_depo[0] * pow(Is, ctl->wet_depo[1]);
01357
01358             /* Calculate exponential decay... */
01359             atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] * lambda);
01360         }
01361     }
01362 }

```

Here is the call graph for this function:



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

Write simulation output.

Definition at line 1366 of file [trac.c](#).

```

01372         {
01373
01374     char filename[2 * LEN];
01375
01376     double r;
01377
01378     int year, mon, day, hour, min, sec, updated = 0;
01379
01380     /* Get time... */
01381     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01382
01383     /* Write atmospheric data... */
01384     if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01385         if (!updated) {
01386             RANGE_PUSH("Update host", NVTX_D2H);
01387 #ifdef _OPENACC
01388             pragma acc update host(atm[:1])
01389 #endif
01390             RANGE_POP;
01391             updated = 1;
01392         }
01393         RANGE_PUSH("Write atm data", NVTX_WRITE);
01394         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01395             dirname, ctl->atm_basename, year, mon, day, hour, min);
01396         write_atm(filename, ctl, atm, t);
01397         RANGE_POP;
01398     }
01399
01400     /* Write gridded data... */
01401     if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01402         if (!updated) {
01403             RANGE_PUSH("Update host", NVTX_D2H);
01404 #ifdef _OPENACC
01405             pragma acc update host(atm[:1])
01406 #endif
01407             RANGE_POP;
01408             updated = 1;
01409         }
01410         RANGE_PUSH("Write grid data", NVTX_WRITE);
01411         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01412             dirname, ctl->grid_basename, year, mon, day, hour, min);
01413         write_grid(filename, ctl, met0, met1, atm, t);
01414         RANGE_POP;
01415     }
01416
01417     /* Write CSI data... */
01418     if (ctl->csi_basename[0] != '-' && fmod(t, ctl->csi_dt_out) == 0) {
01419         if (!updated) {
01420             RANGE_PUSH("Update host", NVTX_D2H);
01421 #ifdef _OPENACC
01422             pragma acc update host(atm[:1])
01423 #endif
01424             RANGE_POP;
01425             updated = 1;
01426         }
01427         RANGE_PUSH("Write CSI data", NVTX_WRITE);
01428         sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01429         write_csi(filename, ctl, atm, t);
01430         RANGE_POP;
01431     }
01432 }

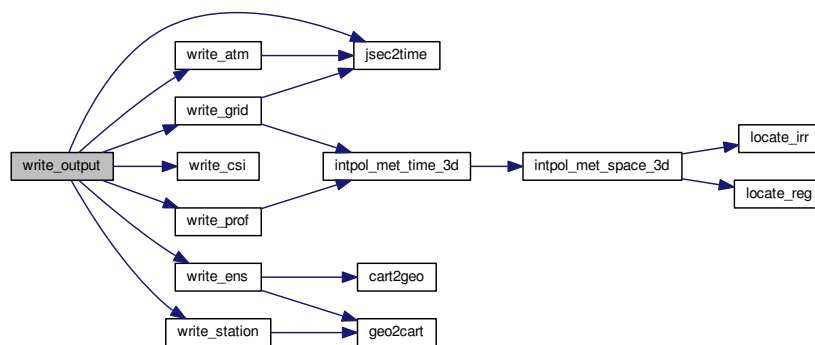
```

```

01431 }
01432
01433 /* Write ensemble data... */
01434 if (ctl->ens_basename[0] != '-') {
01435     if (!updated) {
01436         RANGE_PUSH("Update host", NVTX_D2H);
01437 #ifdef _OPENACC
01438 #pragma acc update host (atm[:1])
01439 #endif
01440         RANGE_POP;
01441         updated = 1;
01442     }
01443     RANGE_PUSH("Write ensemble data", NVTX_WRITE);
01444     sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01445     write_ens(filename, ctl, atm, t);
01446     RANGE_POP;
01447 }
01448
01449 /* Write profile data... */
01450 if (ctl->prof_basename[0] != '-') {
01451     if (!updated) {
01452         RANGE_PUSH("Update host", NVTX_D2H);
01453 #ifdef _OPENACC
01454 #pragma acc update host (atm[:1])
01455 #endif
01456         RANGE_POP;
01457         updated = 1;
01458     }
01459     RANGE_PUSH("Write profile data", NVTX_WRITE);
01460     sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01461     write_prof(filename, ctl, met0, met1, atm, t);
01462     RANGE_POP;
01463 }
01464
01465 /* Write station data... */
01466 if (ctl->stat_basename[0] != '-') {
01467     if (!updated) {
01468         RANGE_PUSH("Update host", NVTX_D2H);
01469 #ifdef _OPENACC
01470 #pragma acc update host (atm[:1])
01471 #endif
01472         RANGE_POP;
01473         updated = 1;
01474     }
01475     RANGE_PUSH("Write station data", NVTX_WRITE);
01476     sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01477     write_station(filename, ctl, atm, t);
01478     RANGE_POP;
01479 }
01480 }

```

Here is the call graph for this function:



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

Definition at line 147 of file [trac.c](#).

```

00149         {
00150
00151     ctl_t ctl;
00152
00153     atm_t *atm;
00154
00155     cache_t *cache;
00156
00157     met_t *met0, *met1;
00158
00159     FILE *dirlist;
00160
00161     char dirname[LEN], filename[2 * LEN];
00162
00163     double *dt, *rs, t;
00164
00165     int num_devices = 0, ntask = -1, rank = 0, size = 1;
00166
00167     /* Initialize MPI... */
00168 #ifdef MPI
00169     RANGE_PUSH("Initialize MPI", NVTX_CPU);
00170     MPI_Init(&argc, &argv);
00171     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00172     MPI_Comm_size(MPI_COMM_WORLD, &size);
00173     RANGE_POP;
00174 #endif
00175
00176     /* Initialize GPUs... */
00177 #ifdef _OPENACC
00178     RANGE_PUSH("Initialize GPUs", NVTX_GPU);
00179     acc_device_t device_type = acc_get_device_type();
00180     num_devices = acc_get_num_devices(acc_device_nvidia);
00181     int device_num = rank % num_devices;
00182     acc_set_device_num(device_num, acc_device_nvidia);
00183     acc_init(device_type);
00184     RANGE_POP;
00185 #endif
00186
00187     /* Check arguments... */
00188     if (argc < 5)
00189         ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00190
00191     /* Open directory list... */
00192     if (!(dirlist = fopen(argv[1], "r")))
00193         ERRMSG("Cannot open directory list!");
00194
00195     /* Loop over directories... */
00196     while (fscanf(dirlist, "%s", dirname) != EOF) {
00197
00198         /* MPI parallelization... */
00199         if ((++ntask) % size != rank)
00200             continue;
00201
00202         /* -----
00203            Initialize model run...
00204            ----- */
00205
00206         /* Set timers... */
00207         START_TIMER(TIMER_TOTAL);
00208         START_TIMER(TIMER_INIT);
00209
00210         /* Allocate... */
00211         RANGE_PUSH("Allocate", NVTX_CPU);
00212         ALLOC(atm, atm_t, 1);
00213         ALLOC(cache, cache_t, 1);
00214         ALLOC(met0, met_t, 1);
00215         ALLOC(met1, met_t, 1);
00216         ALLOC(dt, double,
00217              NP);
00218         ALLOC(rs, double,
00219              3 * NP);
00220         RANGE_POP;
00221
00222         /* Create data region on GPUs... */
00223 #ifdef _OPENACC
00224         RANGE_PUSH("Create data region", NVTX_GPU);
00225         #pragma acc enter data create(atm[:1], cache[:1], ctl, met0[:1], met1[:1], dt[:NP], rs[:3*NP])
00226         RANGE_POP;
00227 #endif
00228
00229         /* Read control parameters... */
00230         RANGE_PUSH("Read ctl", NVTX_READ);
00231         sprintf(filename, "%s/%s", dirname, argv[2]);
00232         read_ctl(filename, argc, argv, &ctl);
00233         RANGE_POP;
00234
00235         /* Read atmospheric data... */

```

```

00236     RANGE_PUSH("Read atm", NVTX_READ);
00237     sprintf(filename, "%s/%s", dirname, argv[3]);
00238     if (!read_atm(filename, &ctl, atm))
00239         ERRMSG("Cannot open file!");
00240     RANGE_POP;
00241
00242     /* Set start time... */
00243     if (ctl.direction == 1) {
00244         ctl.t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00245         if (ctl.t_stop > 1e99)
00246             ctl.t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00247     } else {
00248         ctl.t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00249         if (ctl.t_stop > 1e99)
00250             ctl.t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00251     }
00252
00253     /* Check time interval... */
00254     if (ctl.direction * (ctl.t_stop - ctl.t_start) <= 0)
00255         ERRMSG("Nothing to do!");
00256
00257     /* Round start time... */
00258     if (ctl.direction == 1)
00259         ctl.t_start = floor(ctl.t_start / ctl.dt_mod) * ctl.
dt_mod;
00260     else
00261         ctl.t_start = ceil(ctl.t_start / ctl.dt_mod) * ctl.
dt_mod;
00262
00263     /* Update GPU... */
00264 #ifdef _OPENACC
00265     RANGE_PUSH("Update device", NVTX_H2D);
00266     #pragma acc update device(atm[:l],ctl)
00267     RANGE_POP;
00268 #endif
00269
00270     /* Initialize random number generator... */
00271     module_diffusion_init();
00272
00273     /* Initialize meteorological data... */
00274     RANGE_PUSH("Init meteo data", NVTX_READ);
00275     get_met(&ctl, argv[4], ctl.t_start, &met0, &met1);
00276     if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00277         WARN("Violation of CFL criterion! Check DT_MOD!");
00278     RANGE_POP;
00279
00280     /* Initialize isosurface... */
00281     RANGE_PUSH("Init isosurface...", NVTX_CPU);
00282     if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00283         module_isosurf_init(&ctl, met0, met1, atm, cache);
00284     RANGE_POP;
00285
00286     /* Update GPU... */
00287 #ifdef _OPENACC
00288     RANGE_PUSH("Update device", NVTX_H2D);
00289     #pragma acc update device(cache[:l])
00290     RANGE_POP;
00291 #endif
00292
00293     /* Set timers... */
00294     STOP_TIMER(TIMER_INIT);
00295
00296     /* -----
00297     Loop over timesteps...
00298     ----- */
00299
00300     /* Loop over timesteps... */
00301     for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.
dt_mod;
00302         t += ctl.direction * ctl.dt_mod) {
00303
00304         /* Adjust length of final time step... */
00305         if (ctl.direction * (t - ctl.t_stop) > 0)
00306             t = ctl.t_stop;
00307
00308         /* Set time steps for air parcels... */
00309         RANGE_PUSH("Set time steps", NVTX_GPU);
00310 #ifdef _OPENACC
00311         #pragma acc parallel loop independent gang vector present(ctl,atm,atm->time,dt)
00312         #endif
00313         for (int ip = 0; ip < atm->np; ip++) {
00314             double atmtime = atm->time[ip];
00315             double tstart = ctl.t_start;
00316             double tstop = ctl.t_stop;
00317             int dir = ctl.direction;
00318             if ((dir * (atmtime - tstart) >= 0 && dir * (atmtime - tstop) <= 0
&& dir * (atmtime - t) < 0))
00319

```

```

00320         dt[ip] = t - atmtime;
00321     else
00322         dt[ip] = 0;
00323     }
00324     RANGE_POP;
00325
00326     /* Get meteorological data... */
00327     RANGE_PUSH("Get meteo data", NVTX_READ);
00328     START_TIMER(TIMER_INPUT);
00329     if (t != ctl.t_start)
00330         get_met(&ctl, argv[4], t, &met0, &met1);
00331     STOP_TIMER(TIMER_INPUT);
00332     RANGE_POP;
00333
00334     /* Check initial positions... */
00335     RANGE_PUSH("Check initial positions", NVTX_GPU);
00336     START_TIMER(TIMER_POSITION);
00337     module_position(met0, met1, atm, dt);
00338     STOP_TIMER(TIMER_POSITION);
00339     RANGE_POP;
00340
00341     /* Advection... */
00342     RANGE_PUSH("Advection", NVTX_GPU);
00343     START_TIMER(TIMER_ADVECT);
00344     module_advection(met0, met1, atm, dt);
00345     STOP_TIMER(TIMER_ADVECT);
00346     RANGE_POP;
00347
00348     /* Turbulent diffusion... */
00349     RANGE_PUSH("Turbulent diffusion", NVTX_GPU);
00350     START_TIMER(TIMER_DIFFTURB);
00351     if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
00352         || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0) {
00353         module_diffusion_rng(rs, 3 * (size_t) atm->np);
00354         module_diffusion_turb(&ctl, atm, dt, rs);
00355     }
00356     STOP_TIMER(TIMER_DIFFTURB);
00357     RANGE_POP;
00358
00359     /* Mesoscale diffusion... */
00360     RANGE_PUSH("Mesoscale diffusion", NVTX_GPU);
00361     START_TIMER(TIMER_DIFFMESO);
00362     if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0) {
00363         module_diffusion_rng(rs, 3 * (size_t) atm->np);
00364         module_diffusion_meso(&ctl, met0, met1, atm, cache, dt, rs);
00365     }
00366     STOP_TIMER(TIMER_DIFFMESO);
00367     RANGE_POP;
00368
00369     /* Sedimentation... */
00370     RANGE_PUSH("Sedimentation", NVTX_GPU);
00371     START_TIMER(TIMER_SEDI);
00372     if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0)
00373         module_sedi(&ctl, met0, met1, atm, dt);
00374     STOP_TIMER(TIMER_SEDI);
00375     RANGE_POP;
00376
00377     /* Isosurface... */
00378     RANGE_PUSH("Isosurface", NVTX_GPU);
00379     START_TIMER(TIMER_ISOSURF);
00380     if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00381         module_isosurf(&ctl, met0, met1, atm, cache);
00382     STOP_TIMER(TIMER_ISOSURF);
00383     RANGE_POP;
00384
00385     /* Check final positions... */
00386     RANGE_PUSH("Check final positions", NVTX_GPU);
00387     START_TIMER(TIMER_POSITION);
00388     module_position(met0, met1, atm, dt);
00389     STOP_TIMER(TIMER_POSITION);
00390     RANGE_POP;
00391
00392     /* Interpolate meteorological data... */
00393     RANGE_PUSH("Interpolate meteo data", NVTX_GPU);
00394     START_TIMER(TIMER_METEO);
00395     if (ctl.met_dt_out > 0
00396         && (ctl.met_dt_out < ctl.dt_mod || fmod(t, ctl.
met_dt_out) == 0))
00397         module_meteo(&ctl, met0, met1, atm);
00398     STOP_TIMER(TIMER_METEO);
00399     RANGE_POP;
00400
00401     /* Decay of particle mass... */
00402     RANGE_PUSH("Decay of particle mass", NVTX_GPU);
00403     START_TIMER(TIMER_DECAY);
00404     if (ctl.tdec_trop > 0 && ctl.tdec_strat > 0)
00405         module_decay(&ctl, atm, dt);

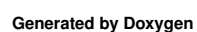
```

```

00406     STOP_TIMER(TIMER_DECAY);
00407     RANGE_POP;
00408
00409     /* OH chemistry... */
00410     RANGE_PUSH("OH chem", NVTX_GPU);
00411     START_TIMER(TIMER_OHCHEM);
00412     if (ctl.oh_chem[0] > 0 && ctl.oh_chem[2] > 0)
00413         module_oh_chem(&ctl, met0, met1, atm, dt);
00414     STOP_TIMER(TIMER_OHCHEM);
00415     RANGE_POP;
00416
00417     /* Wet deposition... */
00418     RANGE_PUSH("Wet deposition", NVTX_GPU);
00419     START_TIMER(TIMER_WETDEPO);
00420     if (ctl.wet_depo[0] > 0 && ctl.wet_depo[1] > 0
00421         && ctl.wet_depo[2] > 0 && ctl.wet_depo[3] > 0)
00422         module_wet_deposition(&ctl, met0, met1, atm, dt);
00423     STOP_TIMER(TIMER_WETDEPO);
00424     RANGE_POP;
00425
00426     /* Write output... */
00427     RANGE_PUSH("Write output", NVTX_WRITE);
00428     START_TIMER(TIMER_OUTPUT);
00429     write_output(dirname, &ctl, met0, met1, atm, t);
00430     STOP_TIMER(TIMER_OUTPUT);
00431     RANGE_POP;
00432 }
00433
00434 /* -----
00435    Finalize model run...
00436    ----- */
00437
00438 /* Report problem size... */
00439 printf("SIZE_NP = %d\n", atm->np);
00440 printf("SIZE_MPI_TASKS = %d\n", size);
00441 printf("SIZE_OMP_THREADS = %d\n", omp_get_max_threads());
00442 printf("SIZE_ACC_DEVICES = %d\n", num_devices);
00443
00444 /* Report memory usage... */
00445 printf("MEMORY_ATM = %g MByte\n", sizeof(atm_t) / 1024. / 1024.);
00446 printf("MEMORY_CACHE = %g MByte\n", sizeof(cache_t) / 1024. / 1024.);
00447 printf("MEMORY_METEO = %g MByte\n", 2 * sizeof(met_t) / 1024. / 1024.);
00448 printf("MEMORY_DYNAMIC = %g MByte\n", (sizeof(met_t)
00449                                         + 4 * NP * sizeof(double)
00450                                         + EX * EY * EP * sizeof(float)) /
00451      1024. / 1024.);
00452 printf("MEMORY_STATIC = %g MByte\n", (EX * EY * sizeof(double)
00453                                         + EX * EY * EP * sizeof(float)
00454                                         + 4 * GX * GY * GZ * sizeof(double)
00455                                         + 2 * GX * GY * GZ * sizeof(int)
00456                                         + 2 * GX * GY * sizeof(double)
00457                                         + GX * GY * sizeof(int)) / 1024. /
00458      1024.);
00459
00460 /* Report timers... */
00461 STOP_TIMER(TIMER_TOTAL);
00462 PRINT_TIMER(TIMER_INIT);
00463 PRINT_TIMER(TIMER_INPUT);
00464 PRINT_TIMER(TIMER_OUTPUT);
00465 PRINT_TIMER(TIMER_ADVECT);
00466 PRINT_TIMER(TIMER_DECAY);
00467 PRINT_TIMER(TIMER_DIFFMESO);
00468 PRINT_TIMER(TIMER_DIFFTURB);
00469 PRINT_TIMER(TIMER_ISOSURF);
00470 PRINT_TIMER(TIMER_METEO);
00471 PRINT_TIMER(TIMER_POSITION);
00472 PRINT_TIMER(TIMER_SEDI);
00473 PRINT_TIMER(TIMER_OHCHEM);
00474 PRINT_TIMER(TIMER_WETDEPO);
00475 PRINT_TIMER(TIMER_TOTAL);
00476
00477 /* Free... */
00478 RANGE_PUSH("Deallocations", NVTX_CPU);
00479 free(atm);
00480 free(cache);
00481 free(met0);
00482 free(met1);
00483 free(dt);
00484 free(rs);
00485 RANGE_POP;
00486
00487 /* Delete data region on GPUs... */
00488 #ifdef _OPENACC
00489     RANGE_PUSH("Delete data region", NVTX_GPU);
00490     #pragma acc exit data delete(ctl, atm, cache, met0, met1, dt, rs)
00491     RANGE_POP;
00492 #endif

```


Here is the call graph for this function:



5.33.3 Variable Documentation

5.33.3.1 static gsl_rng * rng

Definition at line 32 of file [trac.c](#).

5.34 trac.c

```

00001 /*
00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.
00008
00009  MPTRAC is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU General Public License for more details.
00013
00014  You should have received a copy of the GNU General Public License
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2020 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028  Global variables...
00029 ----- */
00030
00031 #ifdef _OPENACC
00032 curandGenerator_t rng;
00033 #else
00034 static gsl_rng *rng[NTHREADS];
00035 #endif
00036
00037 /* -----
00038  Functions...
00039 ----- */
00040
00042 void module_advection(
00043     met_t * met0,
00044     met_t * met1,
00045     atm_t * atm,
00046     double *dt);
00047
00049 void module_decay(
00050     ctl_t * ctl,
00051     atm_t * atm,
00052     double *dt);
00053
00055 void module_diffusion_init(
00056     void);
00057
00059 void module_diffusion_meso(
00060     ctl_t * ctl,
00061     met_t * met0,
00062     met_t * met1,
00063     atm_t * atm,
00064     cache_t * cache,
00065     double *dt,
00066     double *rs);
00067
00069 void module_diffusion_rng(
00070     double *rs,
00071     size_t n);
00072
00074 void module_diffusion_turb(
00075     ctl_t * ctl,
00076     atm_t * atm,
00077     double *dt,
00078     double *rs);
00079
00081 void module_isosurf_init(
00082     ctl_t * ctl,
00083     met_t * met0,
```

```

00084     met_t * met1,
00085     atm_t * atm,
00086     cache_t * cache);
00087
00089 void module_isosurf(
00090     ctl_t * ctl,
00091     met_t * met0,
00092     met_t * met1,
00093     atm_t * atm,
00094     cache_t * cache);
00095
00097 void module_meteo(
00098     ctl_t * ctl,
00099     met_t * met0,
00100     met_t * met1,
00101     atm_t * atm);
00102
00104 void module_position(
00105     met_t * met0,
00106     met_t * met1,
00107     atm_t * atm,
00108     double *dt);
00109
00111 void module_sedi(
00112     ctl_t * ctl,
00113     met_t * met0,
00114     met_t * met1,
00115     atm_t * atm,
00116     double *dt);
00117
00119 void module_oh_chem(
00120     ctl_t * ctl,
00121     met_t * met0,
00122     met_t * met1,
00123     atm_t * atm,
00124     double *dt);
00125
00127 void module_wet_deposition(
00128     ctl_t * ctl,
00129     met_t * met0,
00130     met_t * met1,
00131     atm_t * atm,
00132     double *dt);
00133
00135 void write_output(
00136     const char *dirname,
00137     ctl_t * ctl,
00138     met_t * met0,
00139     met_t * met1,
00140     atm_t * atm,
00141     double t);
00142
00143 /* -----
00144     Main...
00145     ----- */
00146
00147 int main(
00148     int argc,
00149     char *argv[]) {
00150
00151     ctl_t ctl;
00152
00153     atm_t *atm;
00154
00155     cache_t *cache;
00156
00157     met_t *met0, *met1;
00158
00159     FILE *dirlist;
00160
00161     char dirname[LEN], filename[2 * LEN];
00162
00163     double *dt, *rs, t;
00164
00165     int num_devices = 0, ntask = -1, rank = 0, size = 1;
00166
00167     /* Initialize MPI... */
00168 #ifdef MPI
00169     RANGE_PUSH("Initialize MPI", NVTX_CPU);
00170     MPI_Init(&argc, &argv);
00171     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00172     MPI_Comm_size(MPI_COMM_WORLD, &size);
00173     RANGE_POP;
00174 #endif
00175
00176     /* Initialize GPUs... */
00177 #ifdef _OPENACC

```

```

00178 RANGE_PUSH("Initialize GPUs", NVTX_GPU);
00179 acc_device_t device_type = acc_get_device_type();
00180 num_devices = acc_get_num_devices(acc_device_nvidia);
00181 int device_num = rank % num_devices;
00182 acc_set_device_num(device_num, acc_device_nvidia);
00183 acc_init(device_type);
00184 RANGE_POP;
00185 #endif
00186
00187 /* Check arguments... */
00188 if (argc < 5)
00189     ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00190
00191 /* Open directory list... */
00192 if (!(dirlist = fopen(argv[1], "r")))
00193     ERRMSG("Cannot open directory list!");
00194
00195 /* Loop over directories... */
00196 while (fscanf(dirlist, "%s", dirname) != EOF) {
00197
00198     /* MPI parallelization... */
00199     if ((++ntask) % size != rank)
00200         continue;
00201
00202     /* -----
00203        Initialize model run...
00204        ----- */
00205
00206     /* Set timers... */
00207     START_TIMER(TIMER_TOTAL);
00208     START_TIMER(TIMER_INIT);
00209
00210     /* Allocate... */
00211     RANGE_PUSH("Allocate", NVTX_CPU);
00212     ALLOC(atm, atm_t, 1);
00213     ALLOC(cache, cache_t, 1);
00214     ALLOC(met0, met_t, 1);
00215     ALLOC(met1, met_t, 1);
00216     ALLOC(dt, double,
00217           NP);
00218     ALLOC(rs, double,
00219           3 * NP);
00220     RANGE_POP;
00221
00222     /* Create data region on GPUs... */
00223 #ifndef _OPENACC
00224     RANGE_PUSH("Create data region", NVTX_GPU);
00225     #pragma acc enter data create(atm[:1], cache[:1], ctl, met0[:1], met1[:1], dt[:NP], rs[:3*NP])
00226     RANGE_POP;
00227 #endif
00228
00229     /* Read control parameters... */
00230     RANGE_PUSH("Read ctl", NVTX_READ);
00231     sprintf(filename, "%s/%s", dirname, argv[2]);
00232     read_ctl(filename, argc, argv, &ctl);
00233     RANGE_POP;
00234
00235     /* Read atmospheric data... */
00236     RANGE_PUSH("Read atm", NVTX_READ);
00237     sprintf(filename, "%s/%s", dirname, argv[3]);
00238     if (!read_atm(filename, &ctl, atm))
00239         ERRMSG("Cannot open file!");
00240     RANGE_POP;
00241
00242     /* Set start time... */
00243     if (ctl.direction == 1) {
00244         ctl.t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00245         if (ctl.t_stop > 1e99)
00246             ctl.t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00247     } else {
00248         ctl.t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00249         if (ctl.t_stop > 1e99)
00250             ctl.t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00251     }
00252
00253     /* Check time interval... */
00254     if (ctl.direction * (ctl.t_stop - ctl.t_start) <= 0)
00255         ERRMSG("Nothing to do!");
00256
00257     /* Round start time... */
00258     if (ctl.direction == 1)
00259         ctl.t_start = floor(ctl.t_start / ctl.dt_mod) * ctl.
dt_mod;
00260     else
00261         ctl.t_start = ceil(ctl.t_start / ctl.dt_mod) * ctl.
dt_mod;
00262

```

```

00263      /* Update GPU... */
00264 #ifdef _OPENACC
00265     RANGE_PUSH("Update device", NVTX_H2D);
00266 #pragma acc update device(atm[:1],ctl)
00267     RANGE_POP;
00268 #endif
00269
00270     /* Initialize random number generator... */
00271     module_diffusion_init();
00272
00273     /* Initialize meteorological data... */
00274     RANGE_PUSH("Init meteo data", NVTX_READ);
00275     get_met(&ctl, argv[4], ctl.t_start, &met0, &met1);
00276     if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00277         WARN("Violation of CFL criterion! Check DT_MOD!");
00278     RANGE_POP;
00279
00280     /* Initialize isosurface... */
00281     RANGE_PUSH("Init isosurface...", NVTX_CPU);
00282     if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00283         module_isosurf_init(&ctl, met0, met1, atm, cache);
00284     RANGE_POP;
00285
00286     /* Update GPU... */
00287 #ifdef _OPENACC
00288     RANGE_PUSH("Update device", NVTX_H2D);
00289 #pragma acc update device(cache[:1])
00290     RANGE_POP;
00291 #endif
00292
00293     /* Set timers... */
00294     STOP_TIMER(TIMER_INIT);
00295
00296     /* -----
00297     Loop over timesteps...
00298     ----- */
00299
00300     /* Loop over timesteps... */
00301     for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.
dt_mod;
00302         t += ctl.direction * ctl.dt_mod) {
00303
00304         /* Adjust length of final time step... */
00305         if (ctl.direction * (t - ctl.t_stop) > 0)
00306             t = ctl.t_stop;
00307
00308         /* Set time steps for air parcels... */
00309         RANGE_PUSH("Set time steps", NVTX_GPU);
00310 #ifdef _OPENACC
00311 #pragma acc parallel loop independent gang vector present(ctl,atm,atm->time,dt)
00312 #endif
00313         for (int ip = 0; ip < atm->np; ip++) {
00314             double atmtime = atm->time[ip];
00315             double tstart = ctl.t_start;
00316             double tstop = ctl.t_stop;
00317             int dir = ctl.direction;
00318             if ((dir * (atmtime - tstart) >= 0 && dir * (atmtime - tstop) <= 0
&& dir * (atmtime - t) < 0))
00319                 dt[ip] = t - atmtime;
00320             else
00321                 dt[ip] = 0;
00322         }
00323         RANGE_POP;
00324
00325         /* Get meteorological data... */
00326         RANGE_PUSH("Get meteo data", NVTX_READ);
00327         START_TIMER(TIMER_INPUT);
00328         if (t != ctl.t_start)
00329             get_met(&ctl, argv[4], t, &met0, &met1);
00330         STOP_TIMER(TIMER_INPUT);
00331         RANGE_POP;
00332
00333         /* Check initial positions... */
00334         RANGE_PUSH("Check initial positions", NVTX_GPU);
00335         START_TIMER(TIMER_POSITION);
00336         module_position(met0, met1, atm, dt);
00337         STOP_TIMER(TIMER_POSITION);
00338         RANGE_POP;
00339
00340         /* Advection... */
00341         RANGE_PUSH("Advection", NVTX_GPU);
00342         START_TIMER(TIMER_ADVECT);
00343         module_advection(met0, met1, atm, dt);
00344         STOP_TIMER(TIMER_ADVECT);
00345         RANGE_POP;
00346
00347         /* Turbulent diffusion... */
00348

```

```

00349     RANGE_PUSH("Turbulent diffusion", NVTX_GPU);
00350     START_TIMER(TIMER_DIFFTURB);
00351     if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
00352         || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0) {
00353         module_diffusion_rng(rs, 3 * (size_t) atm->np);
00354         module_diffusion_turb(&ctl, atm, dt, rs);
00355     }
00356     STOP_TIMER(TIMER_DIFFTURB);
00357     RANGE_POP;
00358
00359     /* Mesoscale diffusion... */
00360     RANGE_PUSH("Mesoscale diffusion", NVTX_GPU);
00361     START_TIMER(TIMER_DIFFMESO);
00362     if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0) {
00363         module_diffusion_rng(rs, 3 * (size_t) atm->np);
00364         module_diffusion_meso(&ctl, met0, met1, atm, cache, dt, rs);
00365     }
00366     STOP_TIMER(TIMER_DIFFMESO);
00367     RANGE_POP;
00368
00369     /* Sedimentation... */
00370     RANGE_PUSH("Sedimentation", NVTX_GPU);
00371     START_TIMER(TIMER_SEDI);
00372     if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0)
00373         module_sedi(&ctl, met0, met1, atm, dt);
00374     STOP_TIMER(TIMER_SEDI);
00375     RANGE_POP;
00376
00377     /* Isosurface... */
00378     RANGE_PUSH("Isosurface", NVTX_GPU);
00379     START_TIMER(TIMER_ISOSURF);
00380     if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00381         module_isosurf(&ctl, met0, met1, atm, cache);
00382     STOP_TIMER(TIMER_ISOSURF);
00383     RANGE_POP;
00384
00385     /* Check final positions... */
00386     RANGE_PUSH("Check final positions", NVTX_GPU);
00387     START_TIMER(TIMER_POSITION);
00388     module_position(met0, met1, atm, dt);
00389     STOP_TIMER(TIMER_POSITION);
00390     RANGE_POP;
00391
00392     /* Interpolate meteorological data... */
00393     RANGE_PUSH("Interpolate meteo data", NVTX_GPU);
00394     START_TIMER(TIMER_METEO);
00395     if (ctl.met_dt_out > 0
00396         && (ctl.met_dt_out < ctl.dt_mod || fmod(t, ctl.
met_dt_out) == 0))
00397         module_meteo(&ctl, met0, met1, atm);
00398     STOP_TIMER(TIMER_METEO);
00399     RANGE_POP;
00400
00401     /* Decay of particle mass... */
00402     RANGE_PUSH("Decay of particle mass", NVTX_GPU);
00403     START_TIMER(TIMER_DECAY);
00404     if (ctl.tdec_trop > 0 && ctl.tdec_strat > 0)
00405         module_decay(&ctl, atm, dt);
00406     STOP_TIMER(TIMER_DECAY);
00407     RANGE_POP;
00408
00409     /* OH chemistry... */
00410     RANGE_PUSH("OH chem", NVTX_GPU);
00411     START_TIMER(TIMER_OHCHEM);
00412     if (ctl.oh_chem[0] > 0 && ctl.oh_chem[2] > 0)
00413         module_oh_chem(&ctl, met0, met1, atm, dt);
00414     STOP_TIMER(TIMER_OHCHEM);
00415     RANGE_POP;
00416
00417     /* Wet deposition... */
00418     RANGE_PUSH("Wet deposition", NVTX_GPU);
00419     START_TIMER(TIMER_WETDEPO);
00420     if (ctl.wet_depo[0] > 0 && ctl.wet_depo[1] > 0
00421         && ctl.wet_depo[2] > 0 && ctl.wet_depo[3] > 0)
00422         module_wet_deposition(&ctl, met0, met1, atm, dt);
00423     STOP_TIMER(TIMER_WETDEPO);
00424     RANGE_POP;
00425
00426     /* Write output... */
00427     RANGE_PUSH("Write output", NVTX_WRITE);
00428     START_TIMER(TIMER_OUTPUT);
00429     write_output(dirname, &ctl, met0, met1, atm, t);
00430     STOP_TIMER(TIMER_OUTPUT);
00431     RANGE_POP;
00432 }
00433
00434 /* -----

```

```

00435         Finalize model run...
00436         ----- */
00437
00438     /* Report problem size... */
00439     printf("SIZE_NP = %d\n", atm->np);
00440     printf("SIZE_MPI_TASKS = %d\n", size);
00441     printf("SIZE_OMP_THREADS = %d\n", omp_get_max_threads());
00442     printf("SIZE_ACC_DEVICES = %d\n", num_devices);
00443
00444     /* Report memory usage... */
00445     printf("MEMORY_ATM = %g MByte\n", sizeof(atm_t) / 1024. / 1024.);
00446     printf("MEMORY_CACHE = %g MByte\n", sizeof(cache_t) / 1024. / 1024.);
00447     printf("MEMORY_METEO = %g MByte\n", 2 * sizeof(met_t) / 1024. / 1024.);
00448     printf("MEMORY_DYNAMIC = %g MByte\n", (sizeof(met_t)
00449                                         + 4 * NP * sizeof(double)
00450                                         + EX * EY * EP * sizeof(float)) /
00451         1024. / 1024.);
00452     printf("MEMORY_STATIC = %g MByte\n", (EX * EY * sizeof(double)
00453                                         + EX * EY * EP * sizeof(float)
00454                                         + 4 * GX * GY * GZ * sizeof(double)
00455                                         + 2 * GX * GY * GZ * sizeof(int)
00456                                         + 2 * GX * GY * sizeof(double)
00457                                         + GX * GY * sizeof(int)) / 1024. /
00458         1024.);
00459
00460     /* Report timers... */
00461     STOP_TIMER(TIMER_TOTAL);
00462     PRINT_TIMER(TIMER_INIT);
00463     PRINT_TIMER(TIMER_INPUT);
00464     PRINT_TIMER(TIMER_OUTPUT);
00465     PRINT_TIMER(TIMER_ADVECT);
00466     PRINT_TIMER(TIMER_DECAY);
00467     PRINT_TIMER(TIMER_DIFFMESO);
00468     PRINT_TIMER(TIMER_DIFFTURB);
00469     PRINT_TIMER(TIMER_ISOSURF);
00470     PRINT_TIMER(TIMER_METEO);
00471     PRINT_TIMER(TIMER_POSITION);
00472     PRINT_TIMER(TIMER_SEDI);
00473     PRINT_TIMER(TIMER_OHCHEM);
00474     PRINT_TIMER(TIMER_WETDEPO);
00475     PRINT_TIMER(TIMER_TOTAL);
00476
00477     /* Free... */
00478     RANGE_PUSH("Deallocations", NVTX_CPU);
00479     free(atm);
00480     free(cache);
00481     free(met0);
00482     free(met1);
00483     free(dt);
00484     free(rs);
00485     RANGE_POP;
00486
00487     /* Delete data region on GPUs... */
00488     #ifdef _OPENACC
00489     RANGE_PUSH("Delete data region", NVTX_GPU);
00490     #pragma acc exit data delete(ctl,atm,cache,met0,met1,dt,rs)
00491     RANGE_POP;
00492     #endif
00493 }
00494
00495 /* Finalize MPI... */
00496 #ifdef MPI
00497 RANGE_PUSH("Finalize MPI", NVTX_CPU);
00498 MPI_Finalize();
00499 RANGE_POP;
00500 #endif
00501
00502 return EXIT_SUCCESS;
00503 }
00504
00505 /*****
00506
00507 void module_advection(
00508     met_t * met0,
00509     met_t * met1,
00510     atm_t * atm,
00511     double *dt) {
00512
00513     #ifdef _OPENACC
00514     #pragma acc data present(met0,met1,atm,dt)
00515     #pragma acc parallel loop independent gang vector
00516     #else
00517     #pragma omp parallel for default(shared)
00518     #endif
00519     for (int ip = 0; ip < atm->np; ip++)
00520         if (dt[ip] != 0) {
00521

```

```

00522     int ci[3] = { 0 };
00523
00524     double dtm = 0.0, v[3] = { 0.0 }, xm[3] = {
00525         0.0 };
00526     double cw[3] = { 0.0 };
00527
00528     /* Interpolate meteorological data... */
00529     intpol_met_time_3d(met0, met0->u, met1, met1->u, atm->
time[ip],
00530         atm->p[ip], atm->lon[ip], atm->lat[ip], &v[0], ci,
00531         cw, 1);
00532     intpol_met_time_3d(met0, met0->v, met1, met1->v, atm->
time[ip],
00533         atm->p[ip], atm->lon[ip], atm->lat[ip], &v[1], ci,
00534         cw, 0);
00535     intpol_met_time_3d(met0, met0->w, met1, met1->w, atm->
time[ip],
00536         atm->p[ip], atm->lon[ip], atm->lat[ip], &v[2], ci,
00537         cw, 0);
00538
00539     /* Get position of the mid point... */
00540     dtm = atm->time[ip] + 0.5 * dt[ip];
00541     xm[0] =
00542         atm->lon[ip] + DX2DEG(0.5 * dt[ip] * v[0] / 1000., atm->lat[ip]);
00543     xm[1] = atm->lat[ip] + DY2DEG(0.5 * dt[ip] * v[1] / 1000.);
00544     xm[2] = atm->p[ip] + 0.5 * dt[ip] * v[2];
00545
00546     /* Interpolate meteorological data for mid point... */
00547     intpol_met_time_3d(met0, met0->u, met1, met1->u, dtm, xm[2], xm[0],
00548         xm[1], &v[0], ci, cw, 1);
00549     intpol_met_time_3d(met0, met0->v, met1, met1->v, dtm, xm[2], xm[0],
00550         xm[1], &v[1], ci, cw, 0);
00551     intpol_met_time_3d(met0, met0->w, met1, met1->w, dtm, xm[2], xm[0],
00552         xm[1], &v[2], ci, cw, 0);
00553
00554     /* Save new position... */
00555     atm->time[ip] += dt[ip];
00556     atm->lon[ip] += DX2DEG(dt[ip] * v[0] / 1000., xm[1]);
00557     atm->lat[ip] += DY2DEG(dt[ip] * v[1] / 1000.);
00558     atm->p[ip] += dt[ip] * v[2];
00559 }
00560 }
00561
00562 /*****
00563
00564 void module_decay(
00565     ctl_t * ctl,
00566     atm_t * atm,
00567     double *dt) {
00568
00569     /* Check quantity flags... */
00570     if (ctl->qnt_m < 0)
00571         ERRMSG("Module needs quantity mass!");
00572
00573 #ifdef _OPENACC
00574 #pragma acc data present(ctl,atm,dt)
00575 #pragma acc parallel loop independent gang vector
00576 #else
00577 #pragma omp parallel for default(shared)
00578 #endif
00579     for (int ip = 0; ip < atm->np; ip++)
00580         if (dt[ip] != 0) {
00581
00582             double p0, p1, pt, tdec, w;
00583
00584             /* Get tropopause pressure... */
00585             pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00586
00587             /* Get weighting factor... */
00588             p1 = pt * 0.866877899;
00589             p0 = pt / 0.866877899;
00590             if (atm->p[ip] > p0)
00591                 w = 1;
00592             else if (atm->p[ip] < p1)
00593                 w = 0;
00594             else
00595                 w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00596
00597             /* Set lifetime... */
00598             tdec = w * ctl->tdec_trop + (1 - w) * ctl->tdec_strat;
00599
00600             /* Calculate exponential decay... */
00601             atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] / tdec);
00602         }
00603 }
00604
00605 /*****

```



```

00606
00607 void module_diffusion_init(
00608     void) {
00609
00610     /* Initialize random number generator... */
00611     #ifdef _OPENACC
00612
00613         if (curandCreateGenerator(&rng, CURAND_RNG_PSEUDO_DEFAULT)
00614             != CURAND_STATUS_SUCCESS)
00615             ERRMSG("Cannot create random number generator!");
00616         if (curandSetStream(rng, (cudaStream_t) acc_get_cuda_stream(acc_async_sync))
00617             != CURAND_STATUS_SUCCESS)
00618             ERRMSG("Cannot set stream for random number generator!");
00619
00620     #else
00621
00622         gsl_rng_env_setup();
00623         if (omp_get_max_threads() > NTHREADS)
00624             ERRMSG("Too many threads!");
00625         for (int i = 0; i < NTHREADS; i++) {
00626             rng[i] = gsl_rng_alloc(gsl_rng_default);
00627             gsl_rng_set(rng[i], gsl_rng_default_seed + (long unsigned) i);
00628         }
00629
00630     #endif
00631 }
00632
00633 /*****
00634 void module_diffusion_meso(
00635     ctl_t * ctl,
00636     met_t * met0,
00637     met_t * met1,
00638     atm_t * atm,
00639     cache_t * cache,
00640     double *dt,
00641     double *rs) {
00642
00643     #ifdef _OPENACC
00644     #pragma acc data present(ctl,met0,met1,atm,cache,dt,rs)
00645     #pragma acc parallel loop independent gang vector
00646     #else
00647     #pragma omp parallel for default(shared)
00648     #endif
00649     for (int ip = 0; ip < atm->np; ip++)
00650         if (dt[ip] != 0) {
00651
00652             double u[16], v[16], w[16];
00653
00654             /* Get indices... */
00655             int ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
00656             int iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
00657             int iz = locate_irr(met0->p, met0->np, atm->p[ip]);
00658
00659             /* Caching of wind standard deviations... */
00660             if (cache->tsig[ix][iy][iz] != met0->time) {
00661
00662                 /* Collect local wind data... */
00663                 u[0] = met0->u[ix][iy][iz];
00664                 u[1] = met0->u[ix + 1][iy][iz];
00665                 u[2] = met0->u[ix][iy + 1][iz];
00666                 u[3] = met0->u[ix + 1][iy + 1][iz];
00667                 u[4] = met0->u[ix][iy][iz + 1];
00668                 u[5] = met0->u[ix + 1][iy][iz + 1];
00669                 u[6] = met0->u[ix][iy + 1][iz + 1];
00670                 u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00671
00672                 v[0] = met0->v[ix][iy][iz];
00673                 v[1] = met0->v[ix + 1][iy][iz];
00674                 v[2] = met0->v[ix][iy + 1][iz];
00675                 v[3] = met0->v[ix + 1][iy + 1][iz];
00676                 v[4] = met0->v[ix][iy][iz + 1];
00677                 v[5] = met0->v[ix + 1][iy][iz + 1];
00678                 v[6] = met0->v[ix][iy + 1][iz + 1];
00679                 v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00680
00681                 w[0] = met0->w[ix][iy][iz];
00682                 w[1] = met0->w[ix + 1][iy][iz];
00683                 w[2] = met0->w[ix][iy + 1][iz];
00684                 w[3] = met0->w[ix + 1][iy + 1][iz];
00685                 w[4] = met0->w[ix][iy][iz + 1];
00686                 w[5] = met0->w[ix + 1][iy][iz + 1];
00687                 w[6] = met0->w[ix][iy + 1][iz + 1];
00688                 w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00689
00690                 /* Collect local wind data... */
00691                 u[8] = met1->u[ix][iy][iz];

```

```

00693     u[9] = metl->u[ix + 1][iy][iz];
00694     u[10] = metl->u[ix][iy + 1][iz];
00695     u[11] = metl->u[ix + 1][iy + 1][iz];
00696     u[12] = metl->u[ix][iy][iz + 1];
00697     u[13] = metl->u[ix + 1][iy][iz + 1];
00698     u[14] = metl->u[ix][iy + 1][iz + 1];
00699     u[15] = metl->u[ix + 1][iy + 1][iz + 1];
00700
00701     v[8] = metl->v[ix][iy][iz];
00702     v[9] = metl->v[ix + 1][iy][iz];
00703     v[10] = metl->v[ix][iy + 1][iz];
00704     v[11] = metl->v[ix + 1][iy + 1][iz];
00705     v[12] = metl->v[ix][iy][iz + 1];
00706     v[13] = metl->v[ix + 1][iy][iz + 1];
00707     v[14] = metl->v[ix][iy + 1][iz + 1];
00708     v[15] = metl->v[ix + 1][iy + 1][iz + 1];
00709
00710     w[8] = metl->w[ix][iy][iz];
00711     w[9] = metl->w[ix + 1][iy][iz];
00712     w[10] = metl->w[ix][iy + 1][iz];
00713     w[11] = metl->w[ix + 1][iy + 1][iz];
00714     w[12] = metl->w[ix][iy][iz + 1];
00715     w[13] = metl->w[ix + 1][iy][iz + 1];
00716     w[14] = metl->w[ix][iy + 1][iz + 1];
00717     w[15] = metl->w[ix + 1][iy + 1][iz + 1];
00718
00719     /* Get standard deviations of local wind data... */
00720     cache->usig[ix][iy][iz] = (float) stddev(u, 16);
00721     cache->vsig[ix][iy][iz] = (float) stddev(v, 16);
00722     cache->wsig[ix][iy][iz] = (float) stddev(w, 16);
00723     cache->tsig[ix][iy][iz] = met0->time;
00724 }
00725
00726 /* Set temporal correlations for mesoscale fluctuations... */
00727 double r = 1 - 2 * fabs(dt[ip]) / ctl->dt_met;
00728 double r2 = sqrt(1 - r * r);
00729
00730 /* Calculate horizontal mesoscale wind fluctuations... */
00731 if (ctl->turb_mesox > 0) {
00732     cache->up[ip] = (float)
00733         (r * cache->up[ip]
00734          + r2 * rs[3 * ip] * ctl->turb_mesox * cache->usig[ix][iy][iz]);
00735     atm->lon[ip] += DX2DEG(cache->up[ip] * dt[ip] / 1000., atm->lat[ip]);
00736
00737     cache->vp[ip] = (float)
00738         (r * cache->vp[ip]
00739          + r2 * rs[3 * ip + 1] * ctl->turb_mesox * cache->vsig[ix][iy][iz]);
00740     atm->lat[ip] += DY2DEG(cache->vp[ip] * dt[ip] / 1000.);
00741 }
00742
00743 /* Calculate vertical mesoscale wind fluctuations... */
00744 if (ctl->turb_mesoz > 0) {
00745     cache->wp[ip] = (float)
00746         (r * cache->wp[ip]
00747          + r2 * rs[3 * ip + 2] * ctl->turb_mesoz * cache->wsig[ix][iy][iz]);
00748     atm->p[ip] += cache->wp[ip] * dt[ip];
00749 }
00750 }
00751 }
00752
00753 /*****
00754
00755 void module_diffusion_rng(
00756     double *rs,
00757     size_t n) {
00758
00759 #ifdef _OPENACC
00760
00761 #pragma acc host_data use_device(rs)
00762 {
00763     if (curandGenerateNormalDouble(rng, rs, n, 0.0, 1.0)
00764         != CURAND_STATUS_SUCCESS)
00765         ERRMSG("Cannot create random numbers!");
00766 }
00767 #else
00768
00769 #pragma omp parallel for default(shared)
00770 for (size_t i = 0; i < n; ++i)
00771     rs[i] = gsl_ran_gaussian_ziggurat(rng[omp_get_thread_num()], 1.0);
00772 #endif
00773 }
00774
00775
00776 }
00777
00778 /*****
00779

```

```

00780 void module_diffusion_turb(
00781     ctl_t * ctl,
00782     atm_t * atm,
00783     double *dt,
00784     double *rs) {
00785
00786 #ifdef _OPENACC
00787 #pragma acc data present(ctl,atm,dt,rs)
00788 #pragma acc parallel loop independent gang vector
00789 #else
00790 #pragma omp parallel for default(shared)
00791 #endif
00792     for (int ip = 0; ip < atm->np; ip++)
00793         if (dt[ip] != 0) {
00794
00795             double w;
00796
00797             /* Get tropopause pressure... */
00798             double pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00799
00800             /* Get weighting factor... */
00801             double p1 = pt * 0.866877899;
00802             double p0 = pt / 0.866877899;
00803             if (atm->p[ip] > p0)
00804                 w = 1;
00805             else if (atm->p[ip] < p1)
00806                 w = 0;
00807             else
00808                 w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00809
00810             /* Set diffusivity... */
00811             double dx = w * ctl->turb_dx_trop + (1 - w) * ctl->
turb_dx_strat;
00812             double dz = w * ctl->turb_dz_trop + (1 - w) * ctl->
turb_dz_strat;
00813
00814             /* Horizontal turbulent diffusion... */
00815             if (dx > 0) {
00816                 double sigma = sqrt(2.0 * dx * fabs(dt[ip]));
00817                 atm->lon[ip] += DX2DEG(rs[3 * ip] * sigma / 1000., atm->lat[ip]);
00818                 atm->lat[ip] += DY2DEG(rs[3 * ip + 1] * sigma / 1000.);
00819             }
00820
00821             /* Vertical turbulent diffusion... */
00822             if (dz > 0) {
00823                 double sigma = sqrt(2.0 * dz * fabs(dt[ip]));
00824                 atm->p[ip]
00825                     += DZ2DP(rs[3 * ip + 2] * sigma / 1000., atm->p[ip]);
00826             }
00827         }
00828     }
00829
00830 /*****
00831
00832 void module_isosurf_init(
00833     ctl_t * ctl,
00834     met_t * met0,
00835     met_t * met1,
00836     atm_t * atm,
00837     cache_t * cache) {
00838
00839     FILE *in;
00840
00841     char line[LEN];
00842
00843     double t, cw[3];
00844
00845     int ci[3];
00846
00847     /* Save pressure... */
00848     if (ctl->isosurf == 1)
00849         for (int ip = 0; ip < atm->np; ip++)
00850             cache->iso_var[ip] = atm->p[ip];
00851
00852     /* Save density... */
00853     else if (ctl->isosurf == 2)
00854         for (int ip = 0; ip < atm->np; ip++) {
00855             intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
00856                             atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
00857                             1);
00858             cache->iso_var[ip] = atm->p[ip] / t;
00859         }
00860
00861     /* Save potential temperature... */
00862     else if (ctl->isosurf == 3)
00863         for (int ip = 0; ip < atm->np; ip++) {

```

```

00864     intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
00865                     atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
00866                     1);
00867     cache->iso_var[ip] = THETA(atm->p[ip], t);
00868 }
00869
00870 /* Read balloon pressure data... */
00871 else if (ctl->isosurf == 4) {
00872
00873     /* Write info... */
00874     printf("Read balloon pressure data: %s\n", ctl->balloon);
00875
00876     /* Open file... */
00877     if (!(in = fopen(ctl->balloon, "r")))
00878         ERRMSG("Cannot open file!");
00879
00880     /* Read pressure time series... */
00881     while (fgets(line, LEN, in))
00882         if (sscanf(line, "%lg %lg", &(cache->iso_ts[cache->iso_n]),
00883                 &(cache->iso_ps[cache->iso_n])) == 2)
00884             if (++cache->iso_n > NP)
00885                 ERRMSG("Too many data points!");
00886
00887     /* Check number of points... */
00888     if (cache->iso_n < 1)
00889         ERRMSG("Could not read any data!");
00890
00891     /* Close file... */
00892     fclose(in);
00893 }
00894 }
00895
00896 /*****
00897 void module_isosurf(
00898     ctl_t * ctl,
00899     met_t * met0,
00900     met_t * met1,
00901     atm_t * atm,
00902     cache_t * cache) {
00903
00904 #ifdef _OPENACC
00905 #pragma acc data present(ctl,met0,met1,atm,cache)
00906 #pragma acc parallel loop independent gang vector
00907 #else
00908 #pragma omp parallel for default(shared)
00909 #endif
00910     for (int ip = 0; ip < atm->np; ip++) {
00911
00912         double t, cw[3];
00913
00914         int ci[3];
00915
00916         /* Restore pressure... */
00917         if (ctl->isosurf == 1)
00918             atm->p[ip] = cache->iso_var[ip];
00919
00920         /* Restore density... */
00921         else if (ctl->isosurf == 2) {
00922             intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
00923                             atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
00924                             1);
00925             atm->p[ip] = cache->iso_var[ip] * t;
00926         }
00927
00928         /* Restore potential temperature... */
00929         else if (ctl->isosurf == 3) {
00930             intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
00931                             atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
00932                             1);
00933             atm->p[ip] = 1000. * pow(cache->iso_var[ip] / t, -1. / 0.286);
00934         }
00935
00936         /* Interpolate pressure... */
00937         else if (ctl->isosurf == 4) {
00938             if (atm->time[ip] <= cache->iso_ts[0])
00939                 atm->p[ip] = cache->iso_ps[0];
00940             else if (atm->time[ip] >= cache->iso_ts[cache->iso_n - 1])
00941                 atm->p[ip] = cache->iso_ps[cache->iso_n - 1];
00942             else {
00943                 int idx = locate_irr(cache->iso_ts, cache->iso_n, atm->
time[ip]);
00944                 atm->p[ip] = LIN(cache->iso_ts[idx], cache->iso_ps[idx],
cache->iso_ts[idx + 1], cache->iso_ps[idx + 1],

```

```

00947         atm->time[ip]);
00948     }
00949 }
00950 }
00951 }
00952
00953 /*****
00954
00955 void module_meteo(
00956     ctl_t * ctl,
00957     met_t * met0,
00958     met_t * met1,
00959     atm_t * atm) {
00960
00961     /* Check quantity flags... */
00962     if (ctl->qnt_tsts >= 0)
00963         if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
00964             ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
00965
00966 #ifdef _OPENACC
00967 #pragma acc data present(ctl,met0,met1,atm)
00968 #pragma acc parallel loop independent gang vector
00969 #else
00970 #pragma omp parallel for default(shared)
00971 #endif
00972     for (int ip = 0; ip < atm->np; ip++) {
00973
00974         double ps, pt, pc, pv, t, u, v, w, h2o, o3, lwc, iwc, z, cw[3];
00975
00976         int ci[3];
00977
00978         /* Interpolate meteorological data... */
00979         intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->
time[ip],
00980             atm->p[ip], atm->lon[ip], atm->lat[ip], &z, ci, cw, 1);
00981         intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
00982             atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw, 0);
00983         intpol_met_time_3d(met0, met0->u, met1, met1->u, atm->
time[ip],
00984             atm->p[ip], atm->lon[ip], atm->lat[ip], &u, ci, cw, 0);
00985         intpol_met_time_3d(met0, met0->v, met1, met1->v, atm->
time[ip],
00986             atm->p[ip], atm->lon[ip], atm->lat[ip], &v, ci, cw, 0);
00987         intpol_met_time_3d(met0, met0->w, met1, met1->w, atm->
time[ip],
00988             atm->p[ip], atm->lon[ip], atm->lat[ip], &w, ci, cw, 0);
00989         intpol_met_time_3d(met0, met0->pv, met1, met1->pv, atm->
time[ip],
00990             atm->p[ip], atm->lon[ip], atm->lat[ip], &pv, ci, cw,
0);
00991         intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, atm->
time[ip],
00992             atm->p[ip], atm->lon[ip], atm->lat[ip], &h2o, ci, cw,
0);
00993         intpol_met_time_3d(met0, met0->o3, met1, met1->o3, atm->
time[ip],
00994             atm->p[ip], atm->lon[ip], atm->lat[ip], &o3, ci, cw,
0);
00995         intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, atm->
time[ip],
00996             atm->p[ip], atm->lon[ip], atm->lat[ip], &lwc, ci, cw,
0);
00997         intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, atm->
time[ip],
00998             atm->p[ip], atm->lon[ip], atm->lat[ip], &iwc, ci, cw,
0);
00999         intpol_met_time_2d(met0, met0->ps, met1, met1->ps, atm->
time[ip],
01000             atm->lon[ip], atm->lat[ip], &ps, ci, cw, 0);
01001         intpol_met_time_2d(met0, met0->pt, met1, met1->pt, atm->
time[ip],
01002             atm->lon[ip], atm->lat[ip], &pt, ci, cw, 0);
01003         intpol_met_time_2d(met0, met0->pc, met1, met1->pc, atm->
time[ip],
01004             atm->lon[ip], atm->lat[ip], &pc, ci, cw, 0);
01005
01006         /* Set surface pressure... */
01007         if (ctl->qnt_ps >= 0)
01008             atm->q[ctl->qnt_ps][ip] = ps;
01009
01010         /* Set tropopause pressure... */
01011         if (ctl->qnt_pt >= 0)
01012             atm->q[ctl->qnt_pt][ip] = pt;
01013
01014         /* Set pressure... */
01015         if (ctl->qnt_p >= 0)

```

```

01021     atm->q[ctl->qnt_p][ip] = atm->p[ip];
01022
01023     /* Set geopotential height... */
01024     if (ctl->qnt_z >= 0)
01025         atm->q[ctl->qnt_z][ip] = z;
01026
01027     /* Set temperature... */
01028     if (ctl->qnt_t >= 0)
01029         atm->q[ctl->qnt_t][ip] = t;
01030
01031     /* Set zonal wind... */
01032     if (ctl->qnt_u >= 0)
01033         atm->q[ctl->qnt_u][ip] = u;
01034
01035     /* Set meridional wind... */
01036     if (ctl->qnt_v >= 0)
01037         atm->q[ctl->qnt_v][ip] = v;
01038
01039     /* Set vertical velocity... */
01040     if (ctl->qnt_w >= 0)
01041         atm->q[ctl->qnt_w][ip] = w;
01042
01043     /* Set water vapor vmr... */
01044     if (ctl->qnt_h2o >= 0)
01045         atm->q[ctl->qnt_h2o][ip] = h2o;
01046
01047     /* Set ozone vmr... */
01048     if (ctl->qnt_o3 >= 0)
01049         atm->q[ctl->qnt_o3][ip] = o3;
01050
01051     /* Set cloud liquid water content... */
01052     if (ctl->qnt_lwc >= 0)
01053         atm->q[ctl->qnt_lwc][ip] = lwc;
01054
01055     /* Set cloud ice water content... */
01056     if (ctl->qnt_iwc >= 0)
01057         atm->q[ctl->qnt_iwc][ip] = iwc;
01058
01059     /* Set cloud top pressure... */
01060     if (ctl->qnt_pc >= 0)
01061         atm->q[ctl->qnt_pc][ip] = pc;
01062
01063     /* Set nitric acid vmr... */
01064     if (ctl->qnt_hno3 >= 0)
01065         atm->q[ctl->qnt_hno3][ip] =
01066             clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip]);
01067
01068     /* Set hydroxyl number concentration... */
01069     if (ctl->qnt_oh >= 0)
01070         atm->q[ctl->qnt_oh][ip] =
01071             clim_oh(atm->time[ip], atm->lat[ip], atm->p[ip]);
01072
01073     /* Calculate horizontal wind... */
01074     if (ctl->qnt_vh >= 0)
01075         atm->q[ctl->qnt_vh][ip] = sqrt(u * u + v * v);
01076
01077     /* Calculate vertical velocity... */
01078     if (ctl->qnt_vz >= 0)
01079         atm->q[ctl->qnt_vz][ip] = -1e3 * H0 / atm->p[ip] * w;
01080
01081     /* Calculate relative humidity... */
01082     if (ctl->qnt_rh >= 0)
01083         atm->q[ctl->qnt_rh][ip] = RH(atm->p[ip], t, h2o);
01084
01085     /* Calculate potential temperature... */
01086     if (ctl->qnt_theta >= 0)
01087         atm->q[ctl->qnt_theta][ip] = THETA(atm->p[ip], t);
01088
01089     /* Set potential vorticity... */
01090     if (ctl->qnt_pv >= 0)
01091         atm->q[ctl->qnt_pv][ip] = pv;
01092
01093     /* Calculate T_ice (Marti and Mauersberger, 1993)... */
01094     if (ctl->qnt_tice >= 0)
01095         atm->q[ctl->qnt_tice][ip] =
01096             -2663.5 /
01097             (log10((ctl->pssc_h2o > 0 ? ctl->pssc_h2o : h2o) * atm->p[ip] * 100.) -
01098              12.537);
01099
01100     /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
01101     if (ctl->qnt_tnat >= 0) {
01102         double p_hno3;
01103         if (ctl->pssc_hno3 > 0)
01104             p_hno3 = ctl->pssc_hno3 * atm->p[ip] / 1.333224;
01105         else
01106             p_hno3 = clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip])
01107                 * 1e-9 * atm->p[ip] / 1.333224;

```

```

01108     double p_h2o =
01109         (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
01110     double a = 0.009179 - 0.00088 * log10(p_h2o);
01111     double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
01112     double c = -11397.0 / a;
01113     double x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
01114     double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
01115     if (x1 > 0)
01116         atm->q[ctl->qnt_tnat][ip] = x1;
01117     if (x2 > 0)
01118         atm->q[ctl->qnt_tnat][ip] = x2;
01119 }
01120
01121 /* Calculate T_STS (mean of T_ice and T_NAT)... */
01122 if (ctl->qnt_tsts >= 0)
01123     atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
01124                                     + atm->q[ctl->qnt_tnat][ip]);
01125 }
01126 }
01127
01128 /*****
01129 void module_position(
01130     met_t * met0,
01131     met_t * met1,
01132     atm_t * atm,
01133     double *dt) {
01134
01135 #ifdef _OPENACC
01136 #pragma acc data present(met0,met1,atm,dt)
01137 #pragma acc parallel loop independent gang vector
01138 #else
01139 #pragma omp parallel for default(shared)
01140 #endif
01141     for (int ip = 0; ip < atm->np; ip++)
01142         if (dt[ip] != 0) {
01143             double ps, cw[3];
01144             int ci[3];
01145
01146             /* Calculate modulo... */
01147             atm->lon[ip] = FMOD(atm->lon[ip], 360.);
01148             atm->lat[ip] = FMOD(atm->lat[ip], 360.);
01149
01150             /* Check latitude... */
01151             while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
01152                 if (atm->lat[ip] > 90) {
01153                     atm->lat[ip] = 180 - atm->lat[ip];
01154                     atm->lon[ip] += 180;
01155                 }
01156                 if (atm->lat[ip] < -90) {
01157                     atm->lat[ip] = -180 - atm->lat[ip];
01158                     atm->lon[ip] += 180;
01159                 }
01160             }
01161
01162             /* Check longitude... */
01163             while (atm->lon[ip] < -180)
01164                 atm->lon[ip] += 360;
01165             while (atm->lon[ip] >= 180)
01166                 atm->lon[ip] -= 360;
01167
01168             /* Check pressure... */
01169             if (atm->p[ip] < met0->p[met0->np - 1])
01170                 atm->p[ip] = met0->p[met0->np - 1];
01171             else if (atm->p[ip] > 300.) {
01172                 intpol_met_time_2d(met0, met0->ps, met1, met1->ps, atm->
01173 time[ip],
01174                                 atm->lon[ip], atm->lat[ip], &ps, ci, cw, 1);
01175                 if (atm->p[ip] > ps)
01176                     atm->p[ip] = ps;
01177             }
01178         }
01179     }
01180 }
01181
01182 /*****
01183 void module_sedi(
01184     ctl_t * ctl,
01185     met_t * met0,
01186     met_t * met1,
01187     atm_t * atm,
01188     double *dt) {
01189
01190 #ifdef _OPENACC
01191 #pragma acc data present(ctl,met0,met1,atm,dt)

```

```

01194 #pragma acc parallel loop independent gang vector
01195 #else
01196 #pragma omp parallel for default(shared)
01197 #endif
01198     for (int ip = 0; ip < atm->np; ip++)
01199         if (dt[ip] != 0) {
01200             double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p, cw[3];
01201             int ci[3];
01202             /* Convert units... */
01203             p = 100. * atm->p[ip];
01204             r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
01205             rho_p = atm->q[ctl->qnt_rho][ip];
01206             /* Get temperature... */
01207             intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
01208 time[ip],
01209                             atm->p[ip], atm->lon[ip], atm->lat[ip], &T, ci, cw,
01210                             1);
01211             /* Density of dry air... */
01212             rho = p / (RA * T);
01213             /* Dynamic viscosity of air... */
01214             eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
01215             /* Thermal velocity of an air molecule... */
01216             v = sqrt(8. * KB * T / (M_PI * 4.8096e-26));
01217             /* Mean free path of an air molecule... */
01218             lambda = 2. * eta / (rho * v);
01219             /* Knudsen number for air... */
01220             K = lambda / r_p;
01221             /* Cunningham slip-flow correction... */
01222             G = 1. + K * (1.249 + 0.42 * exp(-0.87 / K));
01223             /* Sedimentation (fall) velocity... */
01224             v_p = 2. * SQR(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;
01225             /* Calculate pressure change... */
01226             atm->p[ip] += DZ2DP(v_p * dt[ip] / 1000., atm->p[ip]);
01227         }
01228     }
01229 }
01230
01231 /*****
01232 void module_oh_chem(
01233     ctl_t * ctl,
01234     met_t * met0,
01235     met_t * met1,
01236     atm_t * atm,
01237     double *dt) {
01238     /* Check quantity flags... */
01239     if (ctl->qnt_m < 0)
01240         ERRMSG("Module needs quantity mass!");
01241     #ifdef _OPENACC
01242     #pragma acc data present(ctl,met0,met1,atm,dt)
01243     #pragma acc parallel loop independent gang vector
01244     #else
01245     #pragma omp parallel for default(shared)
01246     #endif
01247     for (int ip = 0; ip < atm->np; ip++)
01248         if (dt[ip] != 0) {
01249             double c, k, k0, ki, M, T, cw[3];
01250             int ci[3];
01251             /* Get temperature... */
01252             intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
01253 time[ip],
01254                             atm->p[ip], atm->lon[ip], atm->lat[ip], &T, ci, cw,
01255                             1);
01256             /* Calculate molecular density... */
01257             M = 7.243e21 * (atm->p[ip] / P0) / T;
01258             /* Calculate rate coefficient for X + OH + M -> XOH + M
01259             (JPL Publication 15-10) ... */
01260             k0 = ctl->oh_chem[0] *
01261                 (ctl->oh_chem[1] > 0 ? pow(T / 300., -ctl->oh_chem[1]) : 1.);

```



```

01279     ki = ctl->oh_chem[2] *
01280         (ctl->oh_chem[3] > 0 ? pow(T / 300., -ctl->oh_chem[3]) : 1.);
01281     c = log10(k0 * M / ki);
01282     k = k0 * M / (1. + k0 * M / ki) * pow(0.6, 1. / (1. + c * c));
01283
01284     /* Calculate exponential decay... */
01285     atm->q[ctl->qnt_m][ip] *=
01286         exp(-dt[ip] * k * clim_oh(atm->time[ip], atm->lat[ip], atm->
p[ip]));
01287 }
01288 }
01289
01290 /*****
01291
01292 void module_wet_deposition(
01293     ctl_t * ctl,
01294     met_t * met0,
01295     met_t * met1,
01296     atm_t * atm,
01297     double *dt) {
01298
01299     /* Check quantity flags... */
01300     if (ctl->qnt_m < 0)
01301         ERRMSG("Module needs quantity mass!");
01302
01303     #ifdef _OPENACC
01304     #pragma acc data present(ctl,met0,met1,atm,dt)
01305     #pragma acc parallel loop independent gang vector
01306     #else
01307     #pragma omp parallel for default(shared)
01308     #endif
01309     for (int ip = 0; ip < atm->np; ip++)
01310         if (dt[ip] != 0) {
01311
01312             double H, Is, Si, T, cl, lambda, iwc, lwc, pc, cw[3];
01313
01314             int inside, ci[3];
01315
01316             /* Check whether particle is below cloud top... */
01317             intpol_met_time_2d(met0, met0->pc, met1, met1->pc, atm->
time[ip],
01318                             atm->lon[ip], atm->lat[ip], &pc, ci, cw, 1);
01319             if (!isfinite(pc) || atm->p[ip] <= pc)
01320                 continue;
01321
01322             /* Check whether particle is inside or below cloud... */
01323             intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, atm->
time[ip],
01324                             atm->p[ip], atm->lon[ip], atm->lat[ip], &lwc, ci, cw,
1);
01325             intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, atm->
time[ip],
01326                             atm->p[ip], atm->lon[ip], atm->lat[ip], &iwc, ci, cw,
0);
01327             inside = (iwc > 0 || lwc > 0);
01328
01329             /* Estimate precipitation rate (Pisso et al., 2019)... */
01330             intpol_met_time_2d(met0, met0->cl, met1, met1->cl, atm->
time[ip],
01331                             atm->lon[ip], atm->lat[ip], &cl, ci, cw, 0);
01332             Is = pow(2. * cl, 1. / 0.36);
01333             if (Is < 0.01)
01334                 continue;
01335
01336             /* Calculate in-cloud scavenging for gases... */
01337             if (inside) {
01338                 /* Get temperature... */
01339                 intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
01340                                     atm->p[ip], atm->lon[ip], atm->lat[ip], &T, ci, cw,
0);
01341
01342                 /* Get Henry's constant (Sander, 2015)... */
01343                 H = ctl->wet_depo[2] * 101.325
01344                     * exp(ctl->wet_depo[3] * (1. / T - 1. / 298.15));
01345
01346                 /* Get scavenging coefficient (Hertel et al., 1995)... */
01347                 Si = 1. / ((1. - cl) / (H * RI / P0 * T) + cl);
01348                 lambda = 6.2 * Si * Is / 3.6e6;
01349             }
01350
01351             /* Calculate below-cloud scavenging for gases (Pisso et al., 2019)... */
01352             else
01353                 lambda = ctl->wet_depo[0] * pow(Is, ctl->wet_depo[1]);
01354
01355             /* Calculate exponential decay... */

```

```

01360         atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] * lambda);
01361     }
01362 }
01363
01364 /*****
01365
01366 void write_output(
01367     const char *dirname,
01368     ctl_t * ctl,
01369     met_t * met0,
01370     met_t * met1,
01371     atm_t * atm,
01372     double t) {
01373
01374     char filename[2 * LEN];
01375
01376     double r;
01377
01378     int year, mon, day, hour, min, sec, updated = 0;
01379
01380     /* Get time... */
01381     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01382
01383     /* Write atmospheric data... */
01384     if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01385         if (!updated) {
01386             RANGE_PUSH("Update host", NVTX_D2H);
01387 #ifdef _OPENACC
01388             pragma acc update host(atm[:1])
01389 #endif
01390             RANGE_POP;
01391             updated = 1;
01392         }
01393         RANGE_PUSH("Write atm data", NVTX_WRITE);
01394         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.tab",
01395             dirname, ctl->atm_basename, year, mon, day, hour, min);
01396         write_atm(filename, ctl, atm, t);
01397         RANGE_POP;
01398     }
01399
01400     /* Write gridded data... */
01401     if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01402         if (!updated) {
01403             RANGE_PUSH("Update host", NVTX_D2H);
01404 #ifdef _OPENACC
01405             pragma acc update host(atm[:1])
01406 #endif
01407             RANGE_POP;
01408             updated = 1;
01409         }
01410         RANGE_PUSH("Write grid data", NVTX_WRITE);
01411         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01412             dirname, ctl->grid_basename, year, mon, day, hour, min);
01413         write_grid(filename, ctl, met0, met1, atm, t);
01414         RANGE_POP;
01415     }
01416
01417     /* Write CSI data... */
01418     if (ctl->csi_basename[0] != '-' && fmod(t, ctl->csi_dt_out) == 0) {
01419         if (!updated) {
01420             RANGE_PUSH("Update host", NVTX_D2H);
01421 #ifdef _OPENACC
01422             pragma acc update host(atm[:1])
01423 #endif
01424             RANGE_POP;
01425             updated = 1;
01426         }
01427         RANGE_PUSH("Write CSI data", NVTX_WRITE);
01428         sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01429         write_csi(filename, ctl, atm, t);
01430         RANGE_POP;
01431     }
01432
01433     /* Write ensemble data... */
01434     if (ctl->ens_basename[0] != '-' && fmod(t, ctl->ens_dt_out) == 0) {
01435         if (!updated) {
01436             RANGE_PUSH("Update host", NVTX_D2H);
01437 #ifdef _OPENACC
01438             pragma acc update host(atm[:1])
01439 #endif
01440             RANGE_POP;
01441             updated = 1;
01442         }
01443         RANGE_PUSH("Write ensemble data", NVTX_WRITE);
01444         sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01445         write_ens(filename, ctl, atm, t);
01446         RANGE_POP;

```

```

01447     }
01448
01449     /* Write profile data... */
01450     if (ctl->prof_basename[0] != '-') {
01451         if (!updated) {
01452             RANGE_PUSH("Update host", NVTX_D2H);
01453 #ifdef _OPENACC
01454 #pragma acc update host (atm[:1])
01455 #endif
01456             RANGE_POP;
01457             updated = 1;
01458         }
01459         RANGE_PUSH("Write profile data", NVTX_WRITE);
01460         sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01461         write_prof(filename, ctl, met0, met1, atm, t);
01462         RANGE_POP;
01463     }
01464
01465     /* Write station data... */
01466     if (ctl->stat_basename[0] != '-') {
01467         if (!updated) {
01468             RANGE_PUSH("Update host", NVTX_D2H);
01469 #ifdef _OPENACC
01470 #pragma acc update host (atm[:1])
01471 #endif
01472             RANGE_POP;
01473             updated = 1;
01474         }
01475         RANGE_PUSH("Write station data", NVTX_WRITE);
01476         sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01477         write_station(filename, ctl, atm, t);
01478         RANGE_POP;
01479     }
01480 }

```

5.35 tropo.c File Reference

Create tropopause climatology from meteorological data.

Functions

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

5.35.1 Detailed Description

Create tropopause climatology from meteorological data.

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

5.35.2 Function Documentation

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

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

```

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 }

```

5.35.2.2 `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)

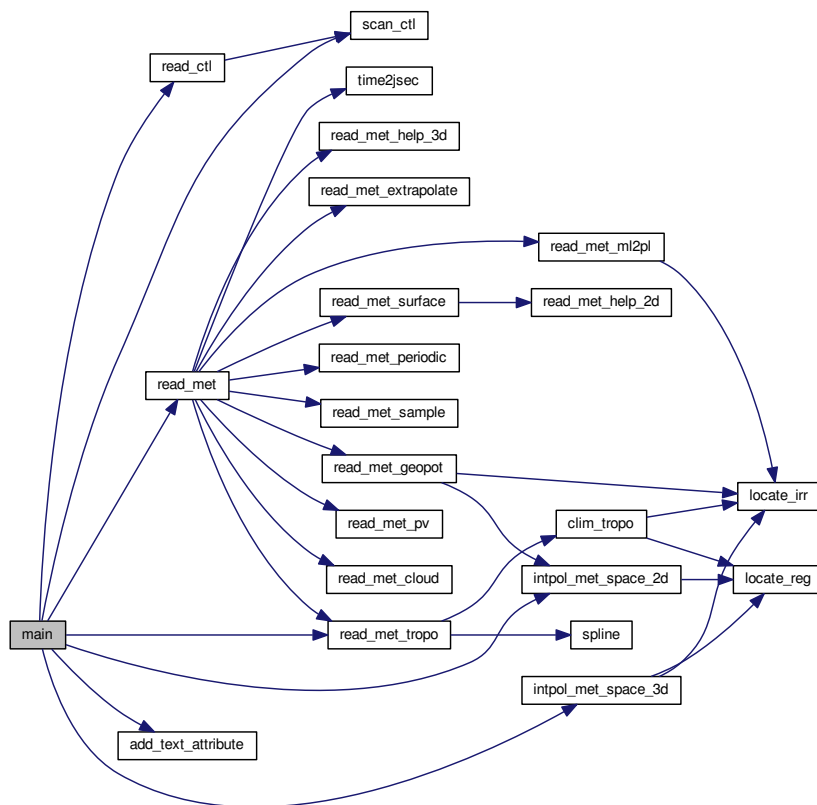
```

```

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:



5.36 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 }

```

5.37 tropo_sample.c File Reference

Sample tropopause climatology.

Functions

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

5.37.1 Detailed Description

Sample tropopause climatology.

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

5.37.2 Function Documentation

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

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

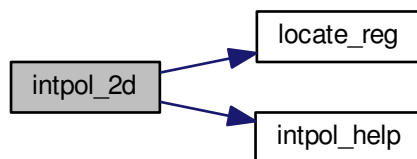
```
00274         {
00275
00276     /* Linear interpolation... */
00277     if (gsl_finite(y0) && gsl_finite(y1))
00278         return LIN(x0, y0, x1, y1, x);
00279
00280     /* Nearest neighbour... */
00281     else {
00282         if (fabs(x - x0) < fabs(x - x1))
00283             return y0;
00284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

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

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

```
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:



5.37.2.3 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, "# $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          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

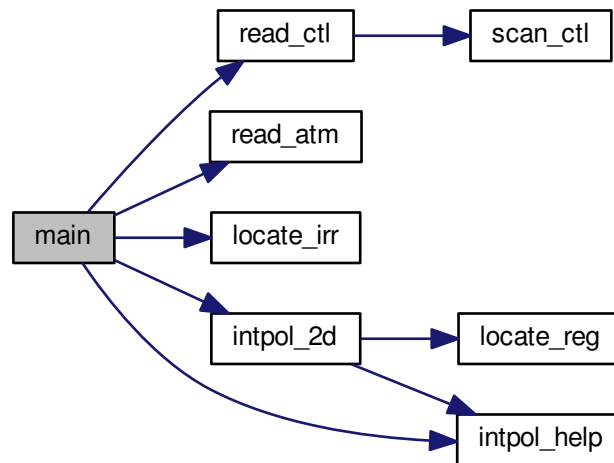
```

```

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:



5.38 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     ctl_t ctl;
00064
00065     atm_t *atm;
00066
00067     static FILE *out;
00068
00069     static char varname[LEN];
00070
00071     static double times[NT], lons[EX], lats[EY], time0, time1, z0, z1, p0, p1,
00072         t0, t1, q0, q1;
00073
00074     static float help[EX * EY], tropo_z0[EX][EY], tropo_z1[EX][EY],
00075         tropo_p0[EX][EY], tropo_p1[EX][EY], tropo_t0[EX][EY],
00076         tropo_t1[EX][EY], tropo_q0[EX][EY], tropo_q1[EX][EY];
00077
00078     static int ip, iq, it, it_old = -999, dimid[10], ncid,
00079         varid, varid_z, varid_p, varid_t, varid_q, h2o;
00080
00081     static size_t count[10], start[10], ntime, nlon, nlat, ilon, ilat;
00082
00083     /* Allocate... */
00084     ALLOC(atm, atm_t, 1);
00085
00086     /* Check arguments... */
00087     if (argc < 5)
00088         ERRMSG("Give parameters: <ctl> <sample.tab> <tropo.nc> <var> <atm_in>");
00089
00090     /* Read control parameters... */
00091     read_ctl(argv[1], argc, argv, &ctl);
00092
00093     /* Read atmospheric data... */
00094     if (!read_atm(argv[5], &ctl, atm))
00095         ERRMSG("Cannot open file!");
00096
00097     /* Open tropopause file... */
00098     printf("Read tropopause data: %s\n", argv[3]);
00099     if (nc_open(argv[3], NC_NOWRITE, &ncid) != NC_NOERR)
00100         ERRMSG("Cannot open file!");
00101
00102     /* Get dimensions... */
00103     NC(nc_inq_dimid(ncid, "time", &dimid[0]));
00104     NC(nc_inq_dimlen(ncid, dimid[0], &ntime));
00105     if (ntime > NT)
00106         ERRMSG("Too many times!");
00107     NC(nc_inq_dimid(ncid, "lat", &dimid[1]));
00108     NC(nc_inq_dimlen(ncid, dimid[1], &nlat));
00109     if (nlat > EY)
00110         ERRMSG("Too many latitudes!");
00111     NC(nc_inq_dimid(ncid, "lon", &dimid[2]));
00112     NC(nc_inq_dimlen(ncid, dimid[2], &nlon));
00113     if (nlon > EX)
00114         ERRMSG("Too many longitudes!");
00115
00116     /* Read coordinates... */
00117     NC(nc_inq_varid(ncid, "time", &varid));
00118     NC(nc_get_var_double(ncid, varid, times));
00119     NC(nc_inq_varid(ncid, "lat", &varid));
00120     NC(nc_get_var_double(ncid, varid, lats));
00121     NC(nc_inq_varid(ncid, "lon", &varid));
00122     NC(nc_get_var_double(ncid, varid, lons));
00123
00124     /* Get variable indices... */
00125     sprintf(varname, "%s_z", argv[4]);
00126     NC(nc_inq_varid(ncid, varname, &varid_z));
00127     sprintf(varname, "%s_p", argv[4]);
00128     NC(nc_inq_varid(ncid, varname, &varid_p));
00129     sprintf(varname, "%s_t", argv[4]);
00130     NC(nc_inq_varid(ncid, varname, &varid_t));
00131     sprintf(varname, "%s_q", argv[4]);
00132     h2o = (nc_inq_varid(ncid, varname, &varid_q) == NC_NOERR);
00133
00134     /* Set dimensions... */
00135     count[0] = 1;
00136     count[1] = nlat;
00137

```

```

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, [279](#)
- 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, [160](#)
- 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, 163
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, 288
intpol_help
 tropo_sample.c, 288
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, 160
 - clim_oh, 161
 - clim_tropo, 162
 - day2doy, 163
 - doy2day, 163
 - geo2cart, 163
 - 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, 215
 - met_prof.c, 221
 - met_sample.c, 227
 - met_zm.c, 232
 - time2jsec.c, 239
 - trac.c, 256
 - tropo.c, 279
 - tropo_sample.c, 289
- 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, 215
- met_np
 - ctl_t, 16
- met_p
 - ctl_t, 16
- met_prof.c, 221
 - main, 221
- met_sample.c, 227
 - main, 227
- 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, [232](#)
 - main, [232](#)
- module_advection
 - trac.c, [241](#)
- module_decay
 - trac.c, [242](#)
- module_diffusion_init
 - trac.c, [242](#)
- module_diffusion_meso
 - trac.c, [243](#)
- module_diffusion_rng
 - trac.c, [245](#)
- module_diffusion_turb
 - trac.c, [245](#)
- module_isosurf
 - trac.c, [247](#)
- module_isosurf_init
 - trac.c, [246](#)
- module_meteo
 - trac.c, [248](#)
- module_oh_chem
 - trac.c, [253](#)
- module_position
 - trac.c, [251](#)
- module_sedi
 - trac.c, [252](#)
- module_wet_deposition
 - trac.c, [253](#)
- molmass
 - ctl_t, [18](#)
- np
 - atm_t, [4](#)
 - met_t, [26](#)
- nq
 - ctl_t, [11](#)
- 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, 239
timer
 libtrac.c, 95
 libtrac.h, 192
trac.c, 240
 main, 256
 module_advection, 241
 module_decay, 242
 module_diffusion_init, 242
 module_diffusion_meso, 243
 module_diffusion_rng, 245
 module_diffusion_turb, 245
 module_isosurf, 247
 module_isosurf_init, 246
 module_meteo, 248
 module_oh_chem, 253
 module_position, 251
 module_sedi, 252
 module_wet_deposition, 253
 rng, 262
 write_output, 255
tropo.c, 279
 add_text_attribute, 279
 main, 279
tropo_sample.c, 288
 intpol_2d, 288
 intpol_help, 288
 main, 289
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, 255
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