

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	3
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	6
4.3.1	Detailed Description	10
4.3.2	Field Documentation	11
4.4	met_t Struct Reference	22
4.4.1	Detailed Description	23
4.4.2	Field Documentation	23

5 File Documentation	26
5.1 atm_conv.c File Reference	26
5.1.1 Detailed Description	26
5.1.2 Function Documentation	26
5.2 atm_conv.c	27
5.3 atm_dist.c File Reference	28
5.3.1 Detailed Description	28
5.3.2 Function Documentation	28
5.4 atm_dist.c	32
5.5 atm_init.c File Reference	36
5.5.1 Detailed Description	37
5.5.2 Function Documentation	37
5.6 atm_init.c	38
5.7 atm_split.c File Reference	40
5.7.1 Detailed Description	40
5.7.2 Function Documentation	40
5.8 atm_split.c	42
5.9 atm_stat.c File Reference	45
5.9.1 Detailed Description	45
5.9.2 Function Documentation	45
5.10 atm_stat.c	48
5.11 day2doy.c File Reference	51
5.11.1 Detailed Description	51
5.11.2 Function Documentation	51
5.12 day2doy.c	52
5.13 doy2day.c File Reference	52
5.13.1 Detailed Description	52
5.13.2 Function Documentation	53
5.14 doy2day.c	53
5.15 extract.c File Reference	54

5.15.1 Detailed Description	54
5.15.2 Function Documentation	54
5.16 extract.c	55
5.17 jsec2time.c File Reference	56
5.17.1 Detailed Description	57
5.17.2 Function Documentation	57
5.18 jsec2time.c	57
5.19 libtrac.c File Reference	58
5.19.1 Detailed Description	60
5.19.2 Function Documentation	60
5.20 libtrac.c	101
5.21 libtrac.h File Reference	139
5.21.1 Detailed Description	140
5.21.2 Function Documentation	141
5.22 libtrac.h	182
5.23 met_map.c File Reference	191
5.23.1 Detailed Description	191
5.23.2 Function Documentation	191
5.24 met_map.c	193
5.25 met_prof.c File Reference	195
5.25.1 Detailed Description	195
5.25.2 Function Documentation	196
5.26 met_prof.c	198
5.27 met_sample.c File Reference	200
5.27.1 Detailed Description	200
5.27.2 Function Documentation	201
5.28 met_sample.c	202
5.29 met_zm.c File Reference	204
5.29.1 Detailed Description	204
5.29.2 Function Documentation	204

5.30 met_zm.c	207
5.31 time2jsec.c File Reference	209
5.31.1 Detailed Description	209
5.31.2 Function Documentation	210
5.32 time2jsec.c	210
5.33 trac.c File Reference	211
5.33.1 Detailed Description	212
5.33.2 Function Documentation	212
5.33.3 Variable Documentation	228
5.34 trac.c	228
5.35 tropo.c File Reference	242
5.35.1 Detailed Description	242
5.35.2 Function Documentation	242
5.36 tropo.c	246
Index	251

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	6

met_t	
Meteorological data	22

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	26
atm_dist.c	
Calculate transport deviations of trajectories	28
atm_init.c	
Create atmospheric data file with initial air parcel positions	36
atm_split.c	
Split air parcels into a larger number of parcels	40
atm_stat.c	
Calculate air parcel statistics	45
day2doy.c	
Convert date to day of year	51
doy2day.c	
Convert day of year to date	52
extract.c	
Extract single trajectory from atmospheric data files	54
jsec2time.c	
Convert Julian seconds to date	56
libtrac.c	
MPTRAC library definitions	58
libtrac.h	
MPTRAC library declarations	139
met_map.c	
Extract map from meteorological data	191
met_prof.c	
Extract vertical profile from meteorological data	195
met_sample.c	
Sample meteorological data at given geolocations	200
met_zm.c	
Extract zonal mean from meteorological data	204
time2jsec.c	
Convert date to Julian seconds	209

trac.c	
Lagrangian particle dispersion model	211
tropo.c	
Create tropopause climatology from meteorological data	242

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 [611](#) of file [libtrac.h](#).

4.1.2 Field Documentation

4.1.2.1 int atm_t::np

Number of air pacels.

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

4.1.2.2 double atm_t::time[NP]

Time [s].

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

4.1.2.3 double atm_t::p[NP]

Pressure [hPa].

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

4.1.2.4 double atm_t::lon[NP]

Longitude [deg].

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

4.1.2.5 double atm_t::lat[NP]

Latitude [deg].

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

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

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

Definition at line 629 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 634 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 637 of file [libtrac.h](#).

4.2.2.2 `float cache_t::vp[NP]`

Meridional wind perturbation [m/s].

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

4.2.2.3 `float cache_t::wp[NP]`

Vertical velocity perturbation [hPa/s].

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

4.2.2.4 `double cache_t::iso_var[NP]`

Isosurface variables.

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

4.2.2.5 `double cache_t::iso_ps[NP]`

Isosurface balloon pressure [hPa].

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

4.2.2.6 `double cache_t::iso_ts[NP]`

Isosurface balloon time [s].

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

4.2.2.7 `int cache_t::iso_n`

Isosurface balloon number of data points.

Definition at line 655 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 658 of file [libtrac.h](#).

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

Cache for zonal wind standard deviations.

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

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

Cache for meridional wind standard deviations.

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

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

Cache for vertical velocity standard deviations.

Definition at line 667 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_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) [m^2/s].*
- double [turb_dx_strat](#)
- *Horizontal turbulent diffusion coefficient (stratosphere) [m^2/s].*
- double [turb_dz_trop](#)
- *Vertical turbulent diffusion coefficient (troposphere) [m^2/s].*
- double [turb_dz_strat](#)
- *Vertical turbulent diffusion coefficient (stratosphere) [m^2/s].*
- double [turb_mesox](#)
- *Horizontal scaling factor for mesoscale wind fluctuations.*
- double [turb_mesoz](#)
- *Vertical scaling factor for mesoscale wind fluctuations.*
- double [molmass](#)
- *Molar mass [g/mol].*
- double [tdec_trop](#)
- *Life time of particles (troposphere) [s].*
- double [tdec_strat](#)
- *Life time of particles (stratosphere) [s].*
- double [psc_h2o](#)
- *H2O volume mixing ratio for PSC analysis.*
- double [psc_hno3](#)
- *HNO3 volume mixing ratio for PSC analysis.*
- char [atm_basename](#) [LEN]
- *Baseline of atmospheric data files.*
- char [atm_gpfile](#) [LEN]
- *Gnuplot file for atmospheric data.*

- double `atm_dt_out`
Time step for atmospheric data output [s].
- int `atm_filter`
Time filter for atmospheric data output (0=no, 1=yes).
- int `atm_stride`
Particle index stride for atmospheric data files.
- int `atm_type`
Type of atmospheric data files (0=ASCII, 1=binary, 2=netCDF).
- char `csi_basename` [LEN]
Baseline of CSI data files.
- double `csi_dt_out`
Time step for CSI data output [s].
- char `csi_obsfile` [LEN]
Observation data file for CSI analysis.
- double `csi_obsmin`
Minimum observation index to trigger detection.
- double `csi_modmin`
Minimum column density to trigger detection [kg/m²].
- int `csi_nz`
Number of altitudes of gridded CSI data.
- double `csi_z0`
Lower altitude of gridded CSI data [km].
- double `csi_z1`
Upper altitude of gridded CSI data [km].
- int `csi_nx`
Number of longitudes of gridded CSI data.
- double `csi_lon0`
Lower longitude of gridded CSI data [deg].
- double `csi_lon1`
Upper longitude of gridded CSI data [deg].
- int `csi_ny`
Number of latitudes of gridded CSI data.
- double `csi_lat0`
Lower latitude of gridded CSI data [deg].
- double `csi_lat1`
Upper latitude of gridded CSI data [deg].
- char `grid_basename` [LEN]
Baseline 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 289 of file [libtrac.h](#).

4.3.2 Field Documentation

4.3.2.1 `int ctl_t::nq`

Number of quantities.

Definition at line 292 of file [libtrac.h](#).

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

Quantity names.

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

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

Quantity units.

Definition at line 298 of file [libtrac.h](#).

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

Quantity output format.

Definition at line 301 of file [libtrac.h](#).

4.3.2.5 `int ctl_t::qnt_ens`

Quantity array index for ensemble IDs.

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

4.3.2.6 `int ctl_t::qnt_m`

Quantity array index for mass.

Definition at line 307 of file [libtrac.h](#).

4.3.2.7 `int ctl_t::qnt_rho`

Quantity array index for particle density.

Definition at line 310 of file [libtrac.h](#).

4.3.2.8 `int ctl_t::qnt_r`

Quantity array index for particle radius.

Definition at line 313 of file [libtrac.h](#).

4.3.2.9 `int ctl_t::qnt_ps`

Quantity array index for surface pressure.

Definition at line 316 of file [libtrac.h](#).

4.3.2.10 `int ctl_t::qnt_pt`

Quantity array index for tropopause pressure.

Definition at line 319 of file [libtrac.h](#).

4.3.2.11 `int ctl_t::qnt_z`

Quantity array index for geopotential height.

Definition at line 322 of file [libtrac.h](#).

4.3.2.12 `int ctl_t::qnt_p`

Quantity array index for pressure.

Definition at line 325 of file [libtrac.h](#).

4.3.2.13 `int ctl_t::qnt_t`

Quantity array index for temperature.

Definition at line 328 of file [libtrac.h](#).

4.3.2.14 `int ctl_t::qnt_u`

Quantity array index for zonal wind.

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

4.3.2.15 `int ctl_t::qnt_v`

Quantity array index for meridional wind.

Definition at line 334 of file [libtrac.h](#).

4.3.2.16 `int ctl_t::qnt_w`

Quantity array index for vertical velocity.

Definition at line 337 of file [libtrac.h](#).

4.3.2.17 `int ctl_t::qnt_h2o`

Quantity array index for water vapor vmr.

Definition at line 340 of file [libtrac.h](#).

4.3.2.18 `int ctl_t::qnt_o3`

Quantity array index for ozone vmr.

Definition at line 343 of file [libtrac.h](#).

4.3.2.19 `int ctl_t::qnt_theta`

Quantity array index for potential temperature.

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

4.3.2.20 `int ctl_t::qnt_vh`

Quantity array index for horizontal wind.

Definition at line 349 of file [libtrac.h](#).

4.3.2.21 `int ctl_t::qnt_vz`

Quantity array index for vertical velocity.

Definition at line 352 of file [libtrac.h](#).

4.3.2.22 `int ctl_t::qnt_pv`

Quantity array index for potential vorticity.

Definition at line 355 of file [libtrac.h](#).

4.3.2.23 `int ctl_t::qnt_tice`

Quantity array index for `T_ice`.

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

4.3.2.24 `int ctl_t::qnt_tsts`

Quantity array index for `T_STS`.

Definition at line 361 of file [libtrac.h](#).

4.3.2.25 `int ctl_t::qnt_tnat`

Quantity array index for `T_NAT`.

Definition at line 364 of file [libtrac.h](#).

4.3.2.26 `int ctl_t::qnt_stat`

Quantity array index for station flag.

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

4.3.2.27 `int ctl_t::direction`

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

Definition at line 370 of file [libtrac.h](#).

4.3.2.28 `double ctl_t::t_start`

Start time of simulation [s].

Definition at line 373 of file [libtrac.h](#).

4.3.2.29 `double ctl_t::t_stop`

Stop time of simulation [s].

Definition at line 376 of file [libtrac.h](#).

4.3.2.30 `double ctl_t::dt_mod`

Time step of simulation [s].

Definition at line 379 of file [libtrac.h](#).

4.3.2.31 `double ctl_t::dt_met`

Time step of meteorological data [s].

Definition at line 382 of file [libtrac.h](#).

4.3.2.32 `int ctl_t::met_dx`

Stride for longitudes.

Definition at line 385 of file [libtrac.h](#).

4.3.2.33 `int ctl_t::met_dy`

Stride for latitudes.

Definition at line 388 of file [libtrac.h](#).

4.3.2.34 `int ctl_t::met_dp`

Stride for pressure levels.

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

4.3.2.35 `int ctl_t::met_sx`

Smoothing for longitudes.

Definition at line 394 of file [libtrac.h](#).

4.3.2.36 `int ctl_t::met_sy`

Smoothing for latitudes.

Definition at line 397 of file [libtrac.h](#).

4.3.2.37 `int ctl_t::met_sp`

Smoothing for pressure levels.

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

4.3.2.38 `int ctl_t::met_np`

Number of target pressure levels.

Definition at line 403 of file [libtrac.h](#).

4.3.2.39 `double ctl_t::met_p[EP]`

Target pressure levels [hPa].

Definition at line 406 of file [libtrac.h](#).

4.3.2.40 `int ctl_t::met_tropo`

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

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

4.3.2.41 `char ctl_t::met_geopot[LEN]`

Surface geopotential data file.

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

4.3.2.42 `double ctl_t::met_dt_out`

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

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

4.3.2.43 `char ctl_t::met_stage[LEN]`

Command to stage meteo data.

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

4.3.2.44 `int ctl_t::isosurf`

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

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

4.3.2.45 `char ctl_t::balloon[LEN]`

Balloon position filename.

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

4.3.2.46 `double ctl_t::turb_dx_trop`

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

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

4.3.2.47 `double ctl_t::turb_dx_strat`

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

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

4.3.2.48 `double ctl_t::turb_dz_trop`

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

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

4.3.2.49 `double ctl_t::turb_dz_strat`

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

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

4.3.2.50 `double ctl_t::turb_mesox`

Horizontal scaling factor for mesoscale wind fluctuations.

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

4.3.2.51 `double ctl_t::turb_mesoz`

Vertical scaling factor for mesoscale wind fluctuations.

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

4.3.2.52 `double ctl_t::molmass`

Molar mass [g/mol].

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

4.3.2.53 `double ctl_t::tdec_trop`

Life time of particles (troposphere) [s].

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

4.3.2.54 `double ctl_t::tdec_strat`

Life time of particles (stratosphere) [s].

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

4.3.2.55 `double ctl_t::psc_h2o`

H2O volume mixing ratio for PSC analysis.

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

4.3.2.56 `double ctl_t::psc_hno3`

HNO3 volume mixing ratio for PSC analysis.

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

4.3.2.57 `char ctl_t::atm_basename[LEN]`

Basename of atmospheric data files.

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

4.3.2.58 `char ctl_t::atm_gpfile[LEN]`

Gnuplot file for atmospheric data.

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

4.3.2.59 `double ctl_t::atm_dt_out`

Time step for atmospheric data output [s].

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

4.3.2.60 `int ctl_t::atm_filter`

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

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

4.3.2.61 `int ctl_t::atm_stride`

Particle index stride for atmospheric data files.

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

4.3.2.62 `int ctl_t::atm_type`

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

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

4.3.2.63 `char ctl_t::csi_basename[LEN]`

Basename of CSI data files.

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

4.3.2.64 `double ctl_t::csi_dt_out`

Time step for CSI data output [s].

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

4.3.2.65 `char ctl_t::csi_obsfile[LEN]`

Observation data file for CSI analysis.

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

4.3.2.66 `double ctl_t::csi_obsmin`

Minimum observation index to trigger detection.

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

4.3.2.67 `double ctl_t::csi_modmin`

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

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

4.3.2.68 `int ctl_t::csi_nz`

Number of altitudes of gridded CSI data.

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

4.3.2.69 `double ctl_t::csi_z0`

Lower altitude of gridded CSI data [km].

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

4.3.2.70 `double ctl_t::csi_z1`

Upper altitude of gridded CSI data [km].

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

4.3.2.71 `int ctl_t::csi_nx`

Number of longitudes of gridded CSI data.

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

4.3.2.72 `double ctl_t::csi_lon0`

Lower longitude of gridded CSI data [deg].

Definition at line 507 of file [libtrac.h](#).

4.3.2.73 `double ctl_t::csi_lon1`

Upper longitude of gridded CSI data [deg].

Definition at line 510 of file [libtrac.h](#).

4.3.2.74 `int ctl_t::csi_ny`

Number of latitudes of gridded CSI data.

Definition at line 513 of file [libtrac.h](#).

4.3.2.75 `double ctl_t::csi_lat0`

Lower latitude of gridded CSI data [deg].

Definition at line 516 of file [libtrac.h](#).

4.3.2.76 `double ctl_t::csi_lat1`

Upper latitude of gridded CSI data [deg].

Definition at line 519 of file [libtrac.h](#).

4.3.2.77 `char ctl_t::grid_basename[LEN]`

Basename of grid data files.

Definition at line 522 of file [libtrac.h](#).

4.3.2.78 `char ctl_t::grid_gfile[LEN]`

Gnuplot file for gridded data.

Definition at line 525 of file [libtrac.h](#).

4.3.2.79 `double ctl_t::grid_dt_out`

Time step for gridded data output [s].

Definition at line 528 of file [libtrac.h](#).

4.3.2.80 `int ctl_t::grid_sparse`

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

Definition at line 531 of file [libtrac.h](#).

4.3.2.81 `int ctl_t::grid_nz`

Number of altitudes of gridded data.

Definition at line 534 of file [libtrac.h](#).

4.3.2.82 `double ctl_t::grid_z0`

Lower altitude of gridded data [km].

Definition at line 537 of file [libtrac.h](#).

4.3.2.83 `double ctl_t::grid_z1`

Upper altitude of gridded data [km].

Definition at line 540 of file [libtrac.h](#).

4.3.2.84 `int ctl_t::grid_nx`

Number of longitudes of gridded data.

Definition at line 543 of file [libtrac.h](#).

4.3.2.85 `double ctl_t::grid_lon0`

Lower longitude of gridded data [deg].

Definition at line 546 of file [libtrac.h](#).

4.3.2.86 `double ctl_t::grid_lon1`

Upper longitude of gridded data [deg].

Definition at line 549 of file [libtrac.h](#).

4.3.2.87 `int ctl_t::grid_ny`

Number of latitudes of gridded data.

Definition at line 552 of file [libtrac.h](#).

4.3.2.88 `double ctl_t::grid_lat0`

Lower latitude of gridded data [deg].

Definition at line 555 of file [libtrac.h](#).

4.3.2.89 `double ctl_t::grid_lat1`

Upper latitude of gridded data [deg].

Definition at line 558 of file [libtrac.h](#).

4.3.2.90 `char ctl_t::prof_basename[LEN]`

Basename for profile output file.

Definition at line 561 of file [libtrac.h](#).

4.3.2.91 `char ctl_t::prof_obsfile[LEN]`

Observation data file for profile output.

Definition at line 564 of file [libtrac.h](#).

4.3.2.92 `int ctl_t::prof_nz`

Number of altitudes of gridded profile data.

Definition at line 567 of file [libtrac.h](#).

4.3.2.93 `double ctl_t::prof_z0`

Lower altitude of gridded profile data [km].

Definition at line 570 of file [libtrac.h](#).

4.3.2.94 `double ctl_t::prof_z1`

Upper altitude of gridded profile data [km].

Definition at line 573 of file [libtrac.h](#).

4.3.2.95 `int ctl_t::prof_nx`

Number of longitudes of gridded profile data.

Definition at line 576 of file [libtrac.h](#).

4.3.2.96 `double ctl_t::prof_lon0`

Lower longitude of gridded profile data [deg].

Definition at line 579 of file [libtrac.h](#).

4.3.2.97 `double ctl_t::prof_lon1`

Upper longitude of gridded profile data [deg].

Definition at line 582 of file [libtrac.h](#).

4.3.2.98 `int ctl_t::prof_ny`

Number of latitudes of gridded profile data.

Definition at line 585 of file [libtrac.h](#).

4.3.2.99 double ctl_t::prof_lat0

Lower latitude of gridded profile data [deg].

Definition at line 588 of file [libtrac.h](#).

4.3.2.100 double ctl_t::prof_lat1

Upper latitude of gridded profile data [deg].

Definition at line 591 of file [libtrac.h](#).

4.3.2.101 char ctl_t::ens_basename[LEN]

Basename of ensemble data file.

Definition at line 594 of file [libtrac.h](#).

4.3.2.102 char ctl_t::stat_basename[LEN]

Basename of station data file.

Definition at line 597 of file [libtrac.h](#).

4.3.2.103 double ctl_t::stat_lon

Longitude of station [deg].

Definition at line 600 of file [libtrac.h](#).

4.3.2.104 double ctl_t::stat_lat

Latitude of station [deg].

Definition at line 603 of file [libtrac.h](#).

4.3.2.105 double ctl_t::stat_r

Search radius around station [km].

Definition at line 606 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].
- double [ps](#) [EX][EY]
Surface pressure [hPa].
- double [pt](#) [EX][EY]
Tropopause pressure [hPa].
- float [z](#) [EX][EY][EP]
Geopotential height [km].
- float [t](#) [EX][EY][EP]
Temperature [K].
- float [u](#) [EX][EY][EP]
Zonal wind [m/s].
- float [v](#) [EX][EY][EP]
Meridional wind [m/s].
- float [w](#) [EX][EY][EP]
Vertical 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 [pl](#) [EX][EY][EP]
Pressure on model levels [hPa].

4.4.1 Detailed Description

Meteorological data.

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

4.4.2 Field Documentation

4.4.2.1 double met_t::time

Time [s].

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

4.4.2.2 int met_t::nx

Number of longitudes.

Definition at line 678 of file [libtrac.h](#).

4.4.2.3 int met_t::ny

Number of latitudes.

Definition at line 681 of file [libtrac.h](#).

4.4.2.4 int met_t::np

Number of pressure levels.

Definition at line 684 of file [libtrac.h](#).

4.4.2.5 double met_t::lon[EX]

Longitude [deg].

Definition at line 687 of file [libtrac.h](#).

4.4.2.6 double met_t::lat[EY]

Latitude [deg].

Definition at line 690 of file [libtrac.h](#).

4.4.2.7 double met_t::p[EP]

Pressure [hPa].

Definition at line 693 of file [libtrac.h](#).

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

Surface pressure [hPa].

Definition at line 696 of file [libtrac.h](#).

4.4.2.9 double met_t::pt[EX][EY]

Tropopause pressure [hPa].

Definition at line 699 of file [libtrac.h](#).

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

Geopotential height [km].

Definition at line 702 of file [libtrac.h](#).

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

Temperature [K].

Definition at line 705 of file [libtrac.h](#).

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

Zonal wind [m/s].

Definition at line 708 of file [libtrac.h](#).

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

Meridional wind [m/s].

Definition at line 711 of file [libtrac.h](#).

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

Vertical wind [hPa/s].

Definition at line 714 of file [libtrac.h](#).

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

Potential vorticity [PVU].

Definition at line 717 of file [libtrac.h](#).

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

Water vapor volume mixing ratio [1].

Definition at line 720 of file [libtrac.h](#).

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

Ozone volume mixing ratio [1].

Definition at line 723 of file [libtrac.h](#).

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

Pressure on model levels [hPa].

Definition at line 726 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[])

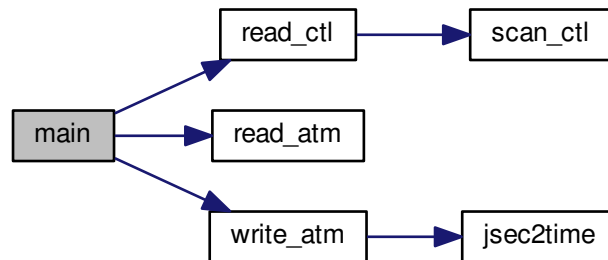
Definition at line 27 of file [atm_conv.c](#).

```

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, pl, *rhtd, *rqtd, *rvtd, rhtdm, rqtdm[NQ], rvtdm,
00042         t, t0 = 0, x0[3], x1[3], x2[3], z1, *z1_old, z2, *z2_old, *work;
00043
00044     int ens, f, init = 0, ip, iq, np, year, mon, day, hour, min;
00045
00046     /* Allocate... */
00047     ALLOC(atm1, atm_t, 1);
00048     ALLOC(atm2, atm_t, 1);
00049     ALLOC(lon1_old, double,
00050         NP);
00051     ALLOC(lat1_old, double,
00052         NP);
00053     ALLOC(z1_old, double,
00054         NP);
00055     ALLOC(lh1, double,
00056         NP);
00057     ALLOC(lv1, double,
00058         NP);
00059     ALLOC(lon2_old, double,
00060         NP);
00061     ALLOC(lat2_old, double,
00062         NP);
00063     ALLOC(z2_old, double,
00064         NP);
00065     ALLOC(lh2, double,
00066         NP);
00067     ALLOC(lv2, double,
00068         NP);
00069     ALLOC(ahtd, double,
00070         NP);
00071     ALLOC(avtd, double,
00072         NP);
00073     ALLOC(aqtd, double,
00074         NP * NQ);
00075     ALLOC(rhtd, double,
00076         NP);

```



```

00077  ALLOC(rvtd, double,
00078      NP);
00079  ALLOC(rqtd, double,
00080      NP * NQ);
00081  ALLOC(work, double,
00082      NP);
00083
00084  /* Check arguments... */
00085  if (argc < 6)
00086      ERRMSG("Give parameters: <ctl> <dist.tab> <param> <atmla> <atmlb>"
00087          " [<atm2a> <atm2b> ...]");
00088
00089  /* Read control parameters... */
00090  read_ctl(argv[1], argc, argv, &ctl);
00091  ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-999", NULL);
00092  p0 = P(scan_ctl(argv[1], argc, argv, "DIST_Z0", -1, "-1000", NULL));
00093  p1 = P(scan_ctl(argv[1], argc, argv, "DIST_Z1", -1, "1000", NULL));
00094  lat0 = scan_ctl(argv[1], argc, argv, "DIST_LAT0", -1, "-1000", NULL);
00095  lat1 = scan_ctl(argv[1], argc, argv, "DIST_LAT1", -1, "1000", NULL);
00096  lon0 = scan_ctl(argv[1], argc, argv, "DIST_LON0", -1, "-1000", NULL);
00097  lon1 = scan_ctl(argv[1], argc, argv, "DIST_LON1", -1, "1000", NULL);
00098
00099  /* Write info... */
00100  printf("Write transport deviations: %s\n", argv[2]);
00101
00102  /* Create output file... */
00103  if (!(out = fopen(argv[2], "w")))
00104      ERRMSG("Cannot create file!");
00105
00106  /* Write header... */
00107  fprintf(out,
00108      "# $1 = time [s]\n"
00109      "# $2 = time difference [s]\n"
00110      "# $3 = absolute horizontal distance (%s) [km]\n"
00111      "# $4 = relative horizontal distance (%s) [%%]\n"
00112      "# $5 = absolute vertical distance (%s) [km]\n"
00113      "# $6 = relative vertical distance (%s) [%%]\n",
00114      argv[3], argv[3], argv[3], argv[3]);
00115  for (iq = 0; iq < ctl.nq; iq++)
00116      fprintf(out,
00117          "# $d = %s absolute difference (%s) [%s]\n"
00118          "# $d = %s relative difference (%s) [%%]\n",
00119          7 + 2 * iq, ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq],
00120          8 + 2 * iq, ctl.qnt_name[iq], argv[3]);
00121  fprintf(out, "# $d = number of particles\n\n", 7 + 2 * ctl.nq);
00122
00123  /* Loop over file pairs... */
00124  for (f = 4; f < argc; f += 2) {
00125
00126      /* Read atmospheric data... */
00127      if (!read_atm(argv[f], &ctl, 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 < atml->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 < atml->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 (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", 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);

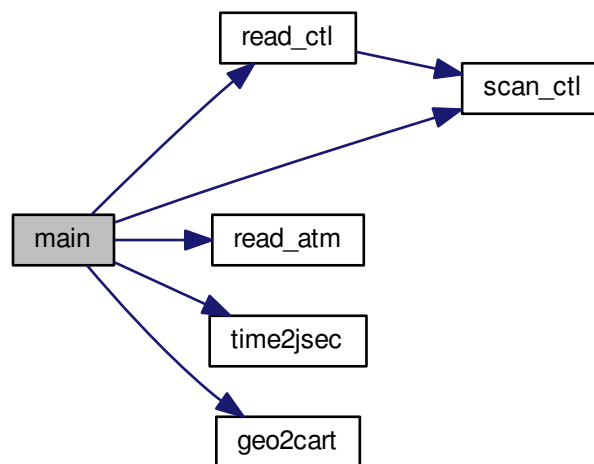
```

```

00338     free(zl_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], 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) [%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 (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     rvtm = 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         rqtmd[iq] = gsl_stats_median(&rqtmd[iq * NP], 1, (size_t) np);
00296     }
00297 } else if (strcasemp(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     rvtm = 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         rqtmd[iq] = gsl_stats_absdev(&rqtmd[iq * NP], 1, (size_t) np);
00305     }
00306 } else if (strcasemp(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     rvtm = 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         rqtmd[iq] = gsl_stats_mad0(&rqtmd[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, rvtm);
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], rqtmd[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(rqtmd);
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, "1", 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));

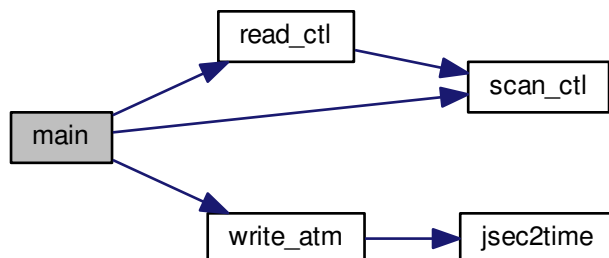
```

```

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, t0);
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, "1", 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

```

```

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, t0);
00122
00123     /* Free... */
00124     gsl_rng_free(rng);
00125     free(atm);
00126
00127     return EXIT_SUCCESS;
00128 }

```

5.7 atm_split.c File Reference

Split air parcels into a larger number of parcels.

Functions

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

5.7.1 Detailed Description

Split air parcels into a larger number of parcels.

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

5.7.2 Function Documentation

5.7.2.1 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;
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     kmax = gsl_stats_max(kk, 1, (size_t) nz);
00100     kmin = gsl_stats_min(kk, 1, (size_t) nz);
00101     for (iz = 0; iz < nz; iz++)
00102         kk[iz] = (kk[iz] - kmin) / (kmax - kmin);
00103 }
00104
00105 /* Get total and maximum mass... */
00106 if (ctl.qnt_m >= 0)
00107     for (ip = 0; ip < atm->np; ip++) {
00108         mtot += atm->q[ctl.qnt_m][ip];
00109         mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
00110     }
00111 if (m > 0)
00112     mtot = m;
00113
00114 /* Loop over air parcels... */
00115 for (i = 0; i < n; i++) {
00116
00117     /* Select air parcel... */
00118     if (ctl.qnt_m >= 0)
00119         do {
00120             ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00121         } while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
00122     else
00123         ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00124
00125     /* Set time... */
00126     if (t1 > t0)
00127         atm2->time[atm2->np] = t0 + (t1 - t0) * gsl_rng_uniform_pos(rng);
00128     else
00129         atm2->time[atm2->np] = atm->time[ip]
00130             + gsl_rng_gaussian_ziggurat(rng, dt / 2.3548);
00131
00132     /* Set vertical position... */
00133     if (nz > 0) {
00134         do {
00135             z = kmin + (kmax - kmin) * gsl_rng_uniform_pos(rng);
00136             iz = locate_irr(kz, nz, z);
00137             k = LIN(kz[iz], kk[iz], kz[iz + 1], kk[iz + 1], z);
00138         } while (gsl_rng_uniform(rng) > k);
00139         atm2->p[atm2->np] = P(z);
00140     }
00141 }

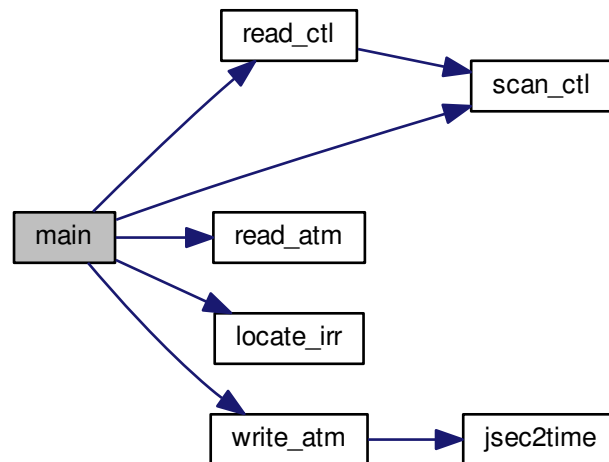
```

```

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

```

Here is the call graph for this function:



5.8 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;
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     kmax = gsl_stats_max(kk, 1, (size_t) nz);
00100     kmin = gsl_stats_min(kk, 1, (size_t) nz);
00101     for (iz = 0; iz < nz; iz++)
00102         kk[iz] = (kk[iz] - kmin) / (kmax - kmin);
00103 }
00104
00105 /* Get total and maximum mass... */
00106 if (ctl.qnt_m >= 0)
00107     for (ip = 0; ip < atm->np; ip++) {
00108         mtot += atm->q[ctl.qnt_m][ip];
00109         mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
00110     }
00111 if (m > 0)
00112     mtot = m;
00113
00114 /* Loop over air parcels... */
00115 for (i = 0; i < n; i++) {
00116
00117     /* Select air parcel... */
00118     if (ctl.qnt_m >= 0)
00119         do {
00120             ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00121             } while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
00122         else
00123             ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00124
00125     /* Set time... */
00126     if (t1 > t0)
00127         atm2->time[atm2->np] = t0 + (t1 - t0) * gsl_rng_uniform_pos(rng);
00128     else
00129         atm2->time[atm2->np] = atm->time[ip]
00130             + gsl_rng_gaussian_ziggurat(rng, dt / 2.3548);
00131
00132     /* Set vertical position... */
00133     if (nz > 0) {
00134         do {
00135             z = kmin + (kmax - kmin) * gsl_rng_uniform_pos(rng);
00136             iz = locate_irr(kz, nz, z);
00137             k = LIN(kz[iz], kk[iz], kz[iz + 1], kk[iz + 1], z);
00138             } while (gsl_rng_uniform(rng) > k);
00139             atm2->p[atm2->np] = P(z);
00140         } else if (z1 > z0)
00141             atm2->p[atm2->np] = P(z0 + (z1 - z0) * gsl_rng_uniform_pos(rng));
00142         else
00143             atm2->p[atm2->np] = atm->p[ip]
00144                 + DZ2DP(gsl_rng_gaussian_ziggurat(rng, dz / 2.3548), atm->p[ip]);
00145
00146     /* Set horizontal position... */
00147     if (lon1 > lon0 && lat1 > lat0) {
00148         atm2->lon[atm2->np] = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
00149         atm2->lat[atm2->np] = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
00150     } else {
00151         atm2->lon[atm2->np] = atm->lon[ip]
00152             + gsl_rng_gaussian_ziggurat(rng, DX2DEG(dx, atm->lat[ip]) / 2.3548);
00153         atm2->lat[atm2->np] = atm->lat[ip]
00154             + gsl_rng_gaussian_ziggurat(rng, DY2DEG(dy) / 2.3548);
00155     }
00156
00157     /* Copy quantities... */
00158     for (iq = 0; iq < ctl.nq; iq++)
00159         atm2->q[iq][atm2->np] = atm->q[iq][ip];
00160
00161     /* Adjust mass... */
00162     if (ctl.qnt_m >= 0)
00163         atm2->q[ctl.qnt_m][atm2->np] = mtot / n;
00164
00165     /* Increment particle counter... */
00166     if ((++atm2->np) > NP)
00167         ERRMSG("Too many air parcels!");
00168 }
00169
00170 /* Save data and close file... */
00171 write_atm(argv[3], &ctl, atm2, atm->time[0]);
00172
00173 /* Free... */
00174 free(atm);
00175 free(atm2);
00176
00177 return EXIT_SUCCESS;
00178 }

```


5.9 atm_stat.c File Reference

Calculate air parcel statistics.

Functions

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

5.9.1 Detailed Description

Calculate air parcel statistics.

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

5.9.2 Function Documentation

5.9.2.1 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> <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 initial 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);

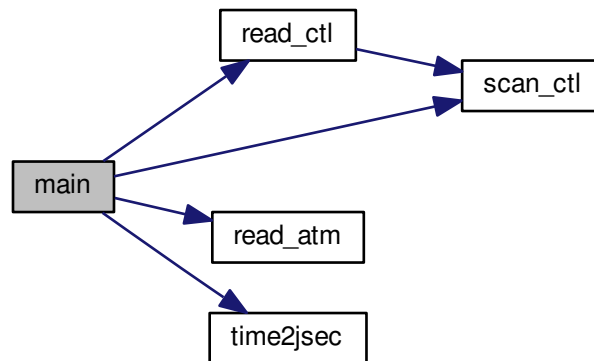
```

```

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 (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:



5.10 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, "# $qd = %s [%s] [%s]\n", iq + 6,
00082         ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq]);
00083 fprintf(out, "# $qd = number of particles\n\n", ctl.nq + 6);
00084
00085 /* Loop over files... */
00086 for (f = 4; f < argc; f++) {
00087
00088     /* Read atmospheric data... */
00089     if (!read_atm(argv[f], &ctl, atm))
00090         continue;
00091
00092     /* Get time from filename... */
00093     sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00094     year = atoi(tstr);
00095     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00096     mon = atoi(tstr);
00097     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00098     day = atoi(tstr);
00099     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00100     hour = atoi(tstr);
00101     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00102     min = atoi(tstr);
00103     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00104
00105     /* Save initial 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 (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 }

```

5.11 day2doy.c File Reference

Convert date to day of year.

Functions

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

5.11.1 Detailed Description

Convert date to day of year.

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

5.11.2 Function Documentation

5.11.2.1 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:



5.12 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 }

```

5.13 doy2day.c File Reference

Convert day of year to date.

Functions

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

5.13.1 Detailed Description

Convert day of year to date.

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

5.13.2 Function Documentation

5.13.2.1 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:



5.14 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
00020 #include "libtrac.h"
00021
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 }

```

5.15 extract.c File Reference

Extract single trajectory from atmospheric data files.

Functions

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

5.15.1 Detailed Description

Extract single trajectory from atmospheric data files.

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

5.15.2 Function Documentation

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

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

```

00029      {
00030
00031      ctl_t ctl;
00032
00033      atm_t *atm;
00034
00035      FILE *out;
00036
00037      int f, ip, iq;
00038
00039      /* Allocate... */
00040      ALLOC(atm, atm_t, 1);
00041
00042      /* Check arguments... */
00043      if (argc < 4)
00044          ERRMSG("Give parameters: <ctl> <trajec.tab> <atm1> [<atm2> ...]");
00045
00046      /* Read control parameters... */
00047      read_ctl(argv[1], argc, argv, &ctl);
00048      ip = (int) scan_ctl(argv[1], argc, argv, "EXTRACT_IP", -1, "", NULL);
00049
00050      /* Write info... */
00051      printf("Write trajectory data: %s\n", argv[2]);
00052
00053      /* Create output file... */
00054      if (!(out = fopen(argv[2], "w")))
00055          ERRMSG("Cannot create file!");
00056
00057      /* Write header... */
00058      fprintf(out,
00059              "# $1 = time [s]\n"
00060              "# $2 = altitude [km]\n"
00061              "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00062      for (iq = 0; iq < ctl.nq; iq++)

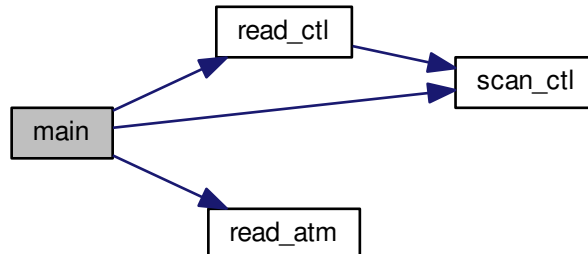
```

```

00063     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00064               ctl.qnt_unit[iq]);
00065     fprintf(out, "\n");
00066
00067     /* Loop over files... */
00068     for (f = 3; f < argc; f++) {
00069
00070         /* Read atmopheric data... */
00071         if (!read_atm(argv[f], &ctl, atm))
00072             continue;
00073
00074         /* Check air parcel index... */
00075         if (ip > atm->np)
00076             ERRMSG("Air parcel index out of range!");
00077
00078         /* Write data... */
00079         fprintf(out, "%.2f %g %g %g", atm->time[ip],
00080               Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
00081         for (iq = 0; iq < ctl.nq; iq++) {
00082             fprintf(out, " ");
00083             fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00084         }
00085         fprintf(out, "\n");
00086     }
00087
00088     /* Close file... */
00089     fclose(out);
00090
00091     /* Free... */
00092     free(atm);
00093
00094     return EXIT_SUCCESS;
00095 }

```

Here is the call graph for this function:



5.16 extract.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     FILE *out;
00036
00037     int f, ip, iq;
00038
00039     /* Allocate... */
00040     ALLOC(atm, atm_t, 1);
00041
00042     /* Check arguments... */
00043     if (argc < 4)
00044         ERRMSG("Give parameters: <ctl> <trajec.tab> <atm1> [<atm2> ...]");
00045
00046     /* Read control parameters... */
00047     read_ctl(argv[1], argc, argv, &ctl);
00048     ip = (int) scan_ctl(argv[1], argc, argv, "EXTRACT_IP", -1, "", NULL);
00049
00050     /* Write info... */
00051     printf("Write trajectory data: %s\n", argv[2]);
00052
00053     /* Create output file... */
00054     if (!(out = fopen(argv[2], "w")))
00055         ERRMSG("Cannot create file!");
00056
00057     /* Write header... */
00058     fprintf(out,
00059         "# $1 = time [s]\n"
00060         "# $2 = altitude [km]\n"
00061         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00062     for (iq = 0; iq < ctl.nq; iq++)
00063         fprintf(out, "# $i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00064             ctl.qnt_unit[iq]);
00065     fprintf(out, "\n");
00066
00067     /* Loop over files... */
00068     for (f = 3; f < argc; f++) {
00069
00070         /* Read atmospheric data... */
00071         if (!read_atm(argv[f], &ctl, atm))
00072             continue;
00073
00074         /* Check air parcel index... */
00075         if (ip > atm->np)
00076             ERRMSG("Air parcel index out of range!");
00077
00078         /* Write data... */
00079         fprintf(out, "%.2f %g %g %g", atm->time[ip],
00080             Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
00081         for (iq = 0; iq < ctl.nq; iq++) {
00082             fprintf(out, " ");
00083             fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00084         }
00085         fprintf(out, "\n");
00086     }
00087
00088     /* Close file... */
00089     fclose(out);
00090
00091     /* Free... */
00092     free(atm);
00093
00094     return EXIT_SUCCESS;
00095 }

```

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\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_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.
- double [intpol_met_2d](#) (double array[EX][EY], int ix, int iy, double wx, double wy)
Linear interpolation of 2-D meteorological data.
- double [intpol_met_3d](#) (float array[EX][EY][EP], int ip, int ix, int iy, double wp, double wx, double wy)
Linear interpolation of 3-D meteorological data.
- void [intpol_met_space](#) ([met_t](#) *met, double p, double lon, double lat, double *ps, double *pt, double *z, double *t, double *u, double *v, double *w, double *pv, double *h2o, double *o3)
Spatial interpolation of meteorological data.
- void [intpol_met_time](#) ([met_t](#) *met0, [met_t](#) *met1, double ts, double p, double lon, double lat, double *ps, double *pt, double *z, double *t, double *u, double *v, double *w, double *pv, double *h2o, double *o3)
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_extrapolate` (`met_t` *met)

Extrapolate meteorological data at lower boundary.
- void `read_met_geopot` (`ctl_t` *ctl, `met_t` *met)

Calculate geopotential heights.
- void `read_met_help` (int ncid, char *varname, char *varname2, `met_t` *met, float dest[EX][EY][EP], float scl)

Read and convert 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_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 }
```

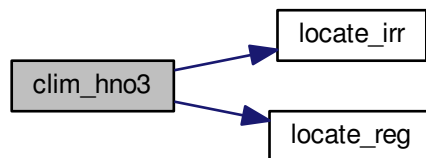
5.19.2.2 double clim_hno3 (double t, double lat, double p)

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
00303     /* Get indices... */
00304     int isec = locate_irr(clim_hno3_secs, 12, sec);
00305     int ilat = locate_reg(clim_hno3_lats, 18, lat);
00306     int ip = locate_irr(clim_hno3_ps, 10, p);
00307
00308     /* Interpolate... */
00309     double aux00 = LIN(clim_hno3_ps[ip], clim_hno3_var[isec][ilat][ip],
00310                       clim_hno3_ps[ip + 1], clim_hno3_var[isec][ilat][ip + 1],
00311                       p);
00312     double aux01 = LIN(clim_hno3_ps[ip], clim_hno3_var[isec][ilat + 1][ip],
00313                       clim_hno3_ps[ip + 1],
00314                       clim_hno3_var[isec][ilat + 1][ip + 1], p);
00315     double aux10 = LIN(clim_hno3_ps[ip], clim_hno3_var[isec + 1][ilat][ip],
00316                       clim_hno3_ps[ip + 1],
00317                       clim_hno3_var[isec + 1][ilat][ip + 1], p);
00318     double aux11 = LIN(clim_hno3_ps[ip], clim_hno3_var[isec + 1][ilat + 1][ip],
00319                       clim_hno3_ps[ip + 1],
00320                       clim_hno3_var[isec + 1][ilat + 1][ip + 1],
00321                       p);
00322     aux00 =
00323         LIN(clim_hno3_lats[ilat], aux00, clim_hno3_lats[ilat + 1], aux01, lat);
00324     aux11 =
00325         LIN(clim_hno3_lats[ilat], aux10, clim_hno3_lats[ilat + 1], aux11, lat);
00326     return LIN(clim_hno3_secs[isec], aux00, clim_hno3_secs[isec + 1], aux11,
00327               sec);
00328 }
```


Here is the call graph for this function:



5.19.2.3 double clim_tropo (double t, double lat)

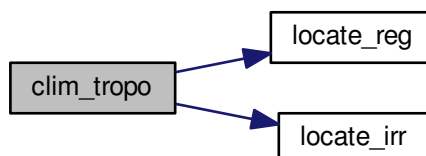
Climatology of tropopause pressure.

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

```

00459         {
00460
00461         /* Get day of year... */
00462         double doy = FMOD(t / 86400., 365.25);
00463         while (doy < 0)
00464             doy += 365.25;
00465
00466         /* Get indices... */
00467         int ilat = locate_reg(clim_tropo_lats, 73, lat);
00468         int imon = locate_irr(clim_tropo_doys, 12, doy);
00469
00470         /* Interpolate... */
00471         double p0 = LIN(clim_tropo_lats[ilat], clim_tropo_tps[imon][ilat],
00472                        clim_tropo_lats[ilat + 1], clim_tropo_tps[imon][ilat + 1],
00473                        lat);
00474         double p1 = LIN(clim_tropo_lats[ilat], clim_tropo_tps[imon + 1][ilat],
00475                        clim_tropo_lats[ilat + 1],
00476                        clim_tropo_tps[imon + 1][ilat + 1],
00477                        lat);
00478         return LIN(clim_tropo_doys[imon], p0, clim_tropo_doys[imon + 1], p1, doy);
00479     }
  
```

Here is the call graph for this function:



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

Get day of year from date.

Definition at line 483 of file libtrac.c.

```
00487         {
00488
00489     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00490     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00491
00492     /* Get day of year... */
00493     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
00494         *doy = d0l[mon - 1] + day - 1;
00495     else
00496         *doy = d0[mon - 1] + day - 1;
00497 }
```

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

Get date from day of year.

Definition at line 501 of file libtrac.c.

```
00505         {
00506
00507     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00508     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00509     int i;
00510
00511     /* Get month and day... */
00512     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
00513         for (i = 11; i >= 0; i--)
00514             if (d0l[i] <= doy)
00515                 break;
00516         *mon = i + 1;
00517         *day = doy - d0l[i] + 1;
00518     } else {
00519         for (i = 11; i >= 0; i--)
00520             if (d0[i] <= doy)
00521                 break;
00522         *mon = i + 1;
00523         *day = doy - d0[i] + 1;
00524     }
00525 }
```

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

Convert geolocation to Cartesian coordinates.

Definition at line 529 of file libtrac.c.

```
00533         {
00534
00535     double radius = z + RE;
00536     x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
00537     x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
00538     x[2] = radius * sin(lat / 180 * M_PI);
00539 }
```

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

Get meteorological data for given timestep.

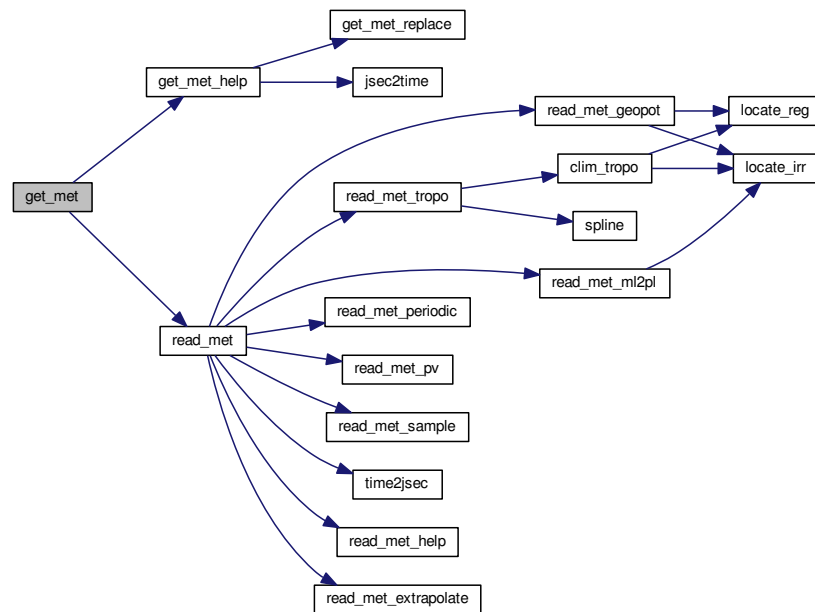
Definition at line 543 of file libtrac.c.

```

00548         {
00549
00550     static int init, ip, ix, iy;
00551
00552     met_t *mets;
00553
00554     char filename[LEN];
00555
00556     /* Init... */
00557     if (t == ctl->t_start || !init) {
00558         init = 1;
00559
00560         get_met_help(t, -1, metabase, ctl->dt_met, filename);
00561         if (!read_met(ctl, filename, *met0))
00562             ERRMSG("Cannot open file!");
00563
00564         get_met_help(t + 1.0 * ctl->direction, 1, metabase, ctl->
dt_met, filename);
00565         if (!read_met(ctl, filename, *met1))
00566             ERRMSG("Cannot open file!");
00567 #ifdef _OPENACC
00568         met_t *met0up = *met0;
00569         met_t *met1up = *met1;
00570 #pragma acc update device(met0up[:1],met1up[:1])
00571 #endif
00572     }
00573
00574     /* Read new data for forward trajectories... */
00575     if (t > (*met1)->time && ctl->direction == 1) {
00576         mets = *met1;
00577         *met1 = *met0;
00578         *met0 = mets;
00579         get_met_help(t, 1, metabase, ctl->dt_met, filename);
00580         if (!read_met(ctl, filename, *met1))
00581             ERRMSG("Cannot open file!");
00582 #ifdef _OPENACC
00583         met_t *met1up = *met1;
00584 #pragma acc update device(met1up[:1])
00585 #endif
00586     }
00587
00588     /* Read new data for backward trajectories... */
00589     if (t < (*met0)->time && ctl->direction == -1) {
00590         mets = *met1;
00591         *met1 = *met0;
00592         *met0 = mets;
00593         get_met_help(t, -1, metabase, ctl->dt_met, filename);
00594         if (!read_met(ctl, filename, *met0))
00595             ERRMSG("Cannot open file!");
00596 #ifdef _OPENACC
00597         met_t *met0up = *met0;
00598 #pragma acc update device(met0up[:1])
00599 #endif
00600     }
00601
00602     /* Check that grids are consistent... */
00603     if (((*met0)->nx != (*met1)->nx
00604         || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
00605         ERRMSG("Meteo grid dimensions do not match!");
00606     for (ix = 0; ix < (*met0)->nx; ix++)
00607         if ((*met0)->lon[ix] != (*met1)->lon[ix])
00608             ERRMSG("Meteo grid longitudes do not match!");
00609     for (iy = 0; iy < (*met0)->ny; iy++)
00610         if ((*met0)->lat[iy] != (*met1)->lat[iy])
00611             ERRMSG("Meteo grid latitudes do not match!");
00612     for (ip = 0; ip < (*met0)->np; ip++)
00613         if ((*met0)->p[ip] != (*met1)->p[ip])
00614             ERRMSG("Meteo grid pressure levels do not match!");
00615 }

```

Here is the call graph for this function:



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

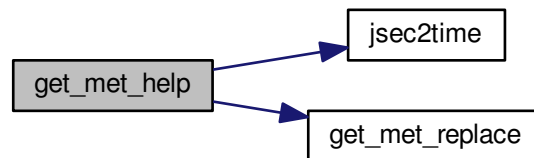
Get meteorological data for timestep.

Definition at line 619 of file libtrac.c.

```

00624         {
00625
00626     char repl[LEN];
00627
00628     double t6, r;
00629
00630     int year, mon, day, hour, min, sec;
00631
00632     /* Round time to fixed intervals... */
00633     if (direct == -1)
00634         t6 = floor(t / dt_met) * dt_met;
00635     else
00636         t6 = ceil(t / dt_met) * dt_met;
00637
00638     /* Decode time... */
00639     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00640
00641     /* Set filename... */
00642     sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
00643     sprintf(repl, "%d", year);
00644     get_met_replace(filename, "YYYY", repl);
00645     sprintf(repl, "%02d", mon);
00646     get_met_replace(filename, "MM", repl);
00647     sprintf(repl, "%02d", day);
00648     get_met_replace(filename, "DD", repl);
00649     sprintf(repl, "%02d", hour);
00650     get_met_replace(filename, "HH", repl);
00651 }
  
```

Here is the call graph for this function:



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

Replace template strings in filename.

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

```

00658         {
00659     char buffer[LEN], *ch;
00660     /* Iterate... */
00661     for (int i = 0; i < 3; i++) {
00662         /* Replace substring... */
00663         if (!(ch = strstr(orig, search)))
00664             return;
00665         strncpy(buffer, orig, (size_t) (ch - orig));
00666         buffer[ch - orig] = 0;
00667         sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
00668         orig[0] = 0;
00669         strcpy(orig, buffer);
00670     }
00671 }
00672
00673 }
00674

```

5.19.2.10 double intpol_met_2d (double array[EX][EY], int ix, int iy, double wx, double wy)

Linear interpolation of 2-D meteorological data.

Definition at line 678 of file [libtrac.c](#).

```

00683     {
00684     /* Set variables... */
00685     double aux00 = array[ix][iy];
00686     double aux01 = array[ix][iy + 1];
00687     double aux10 = array[ix + 1][iy];
00688     double aux11 = array[ix + 1][iy + 1];
00689     /* Interpolate horizontally... */
00690     aux00 = wy * (aux00 - aux01) + aux01;
00691     aux11 = wy * (aux10 - aux11) + aux11;
00692     return wx * (aux00 - aux11) + aux11;
00693 }
00694
00695 }

```

5.19.2.11 double intpol_met_3d (float array[EX][EY][EP], int ip, int ix, int iy, double wp, double wx, double wy)

Linear interpolation of 3-D meteorological data.

Definition at line 699 of file libtrac.c.

```
00706     {
00707
00708     /* Interpolate vertically... */
00709     double aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00710       + array[ix][iy][ip + 1];
00711     double aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
00712       + array[ix][iy + 1][ip + 1];
00713     double aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
00714       + array[ix + 1][iy][ip + 1];
00715     double aux11 =
00716       wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
00717       + array[ix + 1][iy + 1][ip + 1];
00718
00719     /* Interpolate horizontally... */
00720     aux00 = wy * (aux00 - aux01) + aux01;
00721     aux11 = wy * (aux10 - aux11) + aux11;
00722     return wx * (aux00 - aux11) + aux11;
00723 }
```

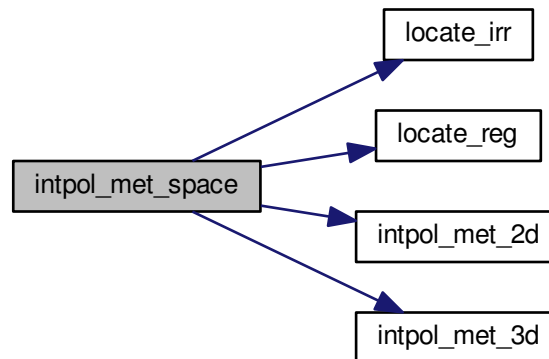
5.19.2.12 void intpol_met_space (met_t * met, double p, double lon, double lat, double * ps, double * pt, double * z, double * t, double * u, double * v, double * w, double * pv, double * h2o, double * o3)

Spatial interpolation of meteorological data.

Definition at line 727 of file libtrac.c.

```
00741     {
00742
00743     /* Check longitude... */
00744     if (met->lon[met->nx - 1] > 180 && lon < 0)
00745         lon += 360;
00746
00747     /* Get indices... */
00748     int ip = locate_irr(met->p, met->np, p);
00749     int ix = locate_reg(met->lon, met->nx, lon);
00750     int iy = locate_reg(met->lat, met->ny, lat);
00751
00752     /* Get weights... */
00753     double wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
00754     double wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
00755     double wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);
00756
00757     /* Interpolate... */
00758     if (ps != NULL)
00759         *ps = intpol_met_2d(met->ps, ix, iy, wx, wy);
00760     if (pt != NULL)
00761         *pt = intpol_met_2d(met->pt, ix, iy, wx, wy);
00762     if (z != NULL)
00763         *z = intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy);
00764     if (t != NULL)
00765         *t = intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy);
00766     if (u != NULL)
00767         *u = intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy);
00768     if (v != NULL)
00769         *v = intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy);
00770     if (w != NULL)
00771         *w = intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy);
00772     if (pv != NULL)
00773         *pv = intpol_met_3d(met->pv, ip, ix, iy, wp, wx, wy);
00774     if (h2o != NULL)
00775         *h2o = intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy);
00776     if (o3 != NULL)
00777         *o3 = intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy);
00778 }
```

Here is the call graph for this function:



5.19.2.13 `void intpol_met_time (met_t * met0, met_t * met1, double ts, double p, double lon, double lat, double * ps, double * pt, double * z, double * t, double * u, double * v, double * w, double * pv, double * h2o, double * o3)`

Temporal interpolation of meteorological data.

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

```

00798     {
00799
00800     double h2o0, h2o1, o30, o31, ps0, ps1, pt0, pt1, pv0, pv1, t0, t1, u0, u1,
00801     v0, v1, w0, w1, wt, z0, z1;
00802
00803     /* Spatial interpolation... */
00804     intpol_met_space(met0, p, lon, lat,
00805                     ps == NULL ? NULL : &ps0,
00806                     pt == NULL ? NULL : &pt0,
00807                     z == NULL ? NULL : &z0,
00808                     t == NULL ? NULL : &t0,
00809                     u == NULL ? NULL : &u0,
00810                     v == NULL ? NULL : &v0,
00811                     w == NULL ? NULL : &w0,
00812                     pv == NULL ? NULL : &pv0,
00813                     h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00814     intpol_met_space(met1, p, lon, lat,
00815                     ps == NULL ? NULL : &ps1,
00816                     pt == NULL ? NULL : &pt1,
00817                     z == NULL ? NULL : &z1,
00818                     t == NULL ? NULL : &t1,
00819                     u == NULL ? NULL : &u1,
00820                     v == NULL ? NULL : &v1,
00821                     w == NULL ? NULL : &w1,
00822                     pv == NULL ? NULL : &pv1,
00823                     h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00824
00825     /* Get weighting factor... */
00826     wt = (met1->time - ts) / (met1->time - met0->time);
00827
00828     /* Interpolate... */
00829     if (ps != NULL)
00830         *ps = wt * (ps0 - ps1) + ps1;
00831     if (pt != NULL)
00832         *pt = wt * (pt0 - pt1) + pt1;
00833     if (z != NULL)
00834         *z = wt * (z0 - z1) + z1;
00835     if (t != NULL)
00836         *t = wt * (t0 - t1) + t1;
00837     if (u != NULL)
00838         *u = wt * (u0 - u1) + u1;

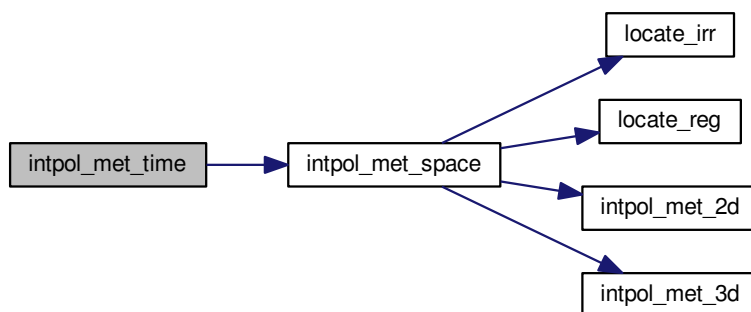
```

```

00839  if (v != NULL)
00840      *v = wt * (v0 - v1) + v1;
00841  if (w != NULL)
00842      *w = wt * (w0 - w1) + w1;
00843  if (pv != NULL)
00844      *pv = wt * (pv0 - pv1) + pv1;
00845  if (h2o != NULL)
00846      *h2o = wt * (h2o0 - h2o1) + h2o1;
00847  if (o3 != NULL)
00848      *o3 = wt * (o30 - o31) + o31;
00849  }

```

Here is the call graph for this function:



5.19.2.14 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 853 of file [libtrac.c](#).

```

00861      {
00862
00863      struct tm t0, *t1;
00864
00865      t0.tm_year = 100;
00866      t0.tm_mon = 0;
00867      t0.tm_mday = 1;
00868      t0.tm_hour = 0;
00869      t0.tm_min = 0;
00870      t0.tm_sec = 0;
00871
00872      time_t jsec0 = (time_t) jsec + timegm(&t0);
00873      t1 = gmtime(&jsec0);
00874
00875      *year = t1->tm_year + 1900;
00876      *mon = t1->tm_mon + 1;
00877      *day = t1->tm_mday;
00878      *hour = t1->tm_hour;
00879      *min = t1->tm_min;
00880      *sec = t1->tm_sec;
00881      *remain = jsec - floor(jsec);
00882  }

```


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

Find array index for irregular grid.

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

```

00889         {
00890
00891     int ilo = 0;
00892     int ihi = n - 1;
00893     int i = (ihi + ilo) >> 1;
00894
00895     if (xx[i] < xx[i + 1])
00896         while (ihi > ilo + 1) {
00897             i = (ihi + ilo) >> 1;
00898             if (xx[i] > x)
00899                 ihi = i;
00900             else
00901                 ilo = i;
00902         } else
00903         while (ihi > ilo + 1) {
00904             i = (ihi + ilo) >> 1;
00905             if (xx[i] <= x)
00906                 ihi = i;
00907             else
00908                 ilo = i;
00909         }
00910
00911     return ilo;
00912 }
```

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

Find array index for regular grid.

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

```

00919         {
00920
00921     /* Calculate index... */
00922     int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
00923
00924     /* Check range... */
00925     if (i < 0)
00926         i = 0;
00927     else if (i >= n - 2)
00928         i = n - 2;
00929
00930     return i;
00931 }
```

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

Read atmospheric data.

Definition at line 935 of file [libtrac.c](#).

```

00938         {
00939
00940     FILE *in;
00941
00942     char line[LEN], *tok;
00943
00944     double t0;
00945
00946     int dimid, ip, iq, ncid, varid;
00947
00948     size_t nparts;
00949
00950     /* Init... */
```

```

00951 atm->np = 0;
00952
00953 /* Write info... */
00954 printf("Read atmospheric data: %s\n", filename);
00955
00956 /* Read ASCII data... */
00957 if (ctl->atm_type == 0) {
00958
00959     /* Open file... */
00960     if (!(in = fopen(filename, "r"))) {
00961         WARN("File not found!");
00962         return 0;
00963     }
00964
00965     /* Read line... */
00966     while (fgets(line, LEN, in)) {
00967
00968         /* Read data... */
00969         TOK(line, tok, "%lg", atm->time[atm->np]);
00970         TOK(NULL, tok, "%lg", atm->p[atm->np]);
00971         TOK(NULL, tok, "%lg", atm->lon[atm->np]);
00972         TOK(NULL, tok, "%lg", atm->lat[atm->np]);
00973         for (iq = 0; iq < ctl->nq; iq++)
00974             TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00975
00976         /* Convert altitude to pressure... */
00977         atm->p[atm->np] = P(atm->p[atm->np]);
00978
00979         /* Increment data point counter... */
00980         if (++atm->np > NP)
00981             ERRMSG("Too many data points!");
00982     }
00983
00984     /* Close file... */
00985     fclose(in);
00986 }
00987
00988 /* Read binary data... */
00989 else if (ctl->atm_type == 1) {
00990
00991     /* Open file... */
00992     if (!(in = fopen(filename, "r")))
00993         return 0;
00994
00995     /* Read data... */
00996     FREAD(&atm->np, int, 1, in);
00997     FREAD(atm->time, double,
00998           (size_t) atm->np,
00999           in);
01000     FREAD(atm->p, double,
01001           (size_t) atm->np,
01002           in);
01003     FREAD(atm->lon, double,
01004           (size_t) atm->np,
01005           in);
01006     FREAD(atm->lat, double,
01007           (size_t) atm->np,
01008           in);
01009     for (iq = 0; iq < ctl->nq; iq++)
01010         FREAD(atm->q[iq], double,
01011               (size_t) atm->np,
01012               in);
01013
01014     /* Close file... */
01015     fclose(in);
01016 }
01017
01018 /* Read netCDF data... */
01019 else if (ctl->atm_type == 2) {
01020
01021     /* Open file... */
01022     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
01023         return 0;
01024
01025     /* Get dimensions... */
01026     NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
01027     NC(nc_inq_dimlen(ncid, dimid, &nparts));
01028     atm->np = (int) nparts;
01029     if (atm->np > NP)
01030         ERRMSG("Too many particles!");
01031
01032     /* Get time... */
01033     NC(nc_inq_varid(ncid, "time", &varid));
01034     NC(nc_get_var_double(ncid, varid, &t0));
01035     for (ip = 0; ip < atm->np; ip++)
01036         atm->time[ip] = t0;
01037

```

```

01038     /* Read geolocations... */
01039     NC(nc_inq_varid(ncid, "PRESS", &varid));
01040     NC(nc_get_var_double(ncid, varid, atm->p));
01041     NC(nc_inq_varid(ncid, "LON", &varid));
01042     NC(nc_get_var_double(ncid, varid, atm->lon));
01043     NC(nc_inq_varid(ncid, "LAT", &varid));
01044     NC(nc_get_var_double(ncid, varid, atm->lat));
01045
01046     /* Read variables... */
01047     if (ctl->qnt_p >= 0)
01048         if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
01049             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
01050     if (ctl->qnt_t >= 0)
01051         if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
01052             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
01053     if (ctl->qnt_u >= 0)
01054         if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
01055             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
01056     if (ctl->qnt_v >= 0)
01057         if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
01058             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
01059     if (ctl->qnt_w >= 0)
01060         if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
01061             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
01062     if (ctl->qnt_h2o >= 0)
01063         if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
01064             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
01065     if (ctl->qnt_o3 >= 0)
01066         if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
01067             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
01068     if (ctl->qnt_theta >= 0)
01069         if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
01070             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
01071     if (ctl->qnt_pv >= 0)
01072         if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
01073             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
01074
01075     /* Check data... */
01076     for (ip = 0; ip < atm->np; ip++)
01077         if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
01078             || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
01079             || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
01080             || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
01081             || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
01082         atm->time[ip] = GSL_NAN;
01083         atm->p[ip] = GSL_NAN;
01084         atm->lon[ip] = GSL_NAN;
01085         atm->lat[ip] = GSL_NAN;
01086         for (iq = 0; iq < ctl->nq; iq++)
01087             atm->q[iq][ip] = GSL_NAN;
01088     } else {
01089         if (ctl->qnt_h2o >= 0)
01090             atm->q[ctl->qnt_h2o][ip] *= 1.608;
01091         if (ctl->qnt_pv >= 0)
01092             atm->q[ctl->qnt_pv][ip] *= 1e6;
01093         if (atm->lon[ip] > 180)
01094             atm->lon[ip] -= 360;
01095     }
01096
01097     /* Close file... */
01098     NC(nc_close(ncid));
01099 }
01100
01101 /* Error... */
01102 else
01103     ERRMSG("Atmospheric data type not supported!");
01104
01105 /* Check number of points... */
01106 if (atm->np < 1)
01107     ERRMSG("Can not read any data!");
01108
01109 /* Return success... */
01110 return 1;
01111 }

```

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

Read control parameters.

Definition at line 1115 of file libtrac.c.

```

01119         {
01120
01121         /* Write info... */
01122         printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
01123             "(executable: %s | compiled: %s, %s)\n\n",
01124             argv[0], __DATE__, __TIME__);
01125
01126         /* Initialize quantity indices... */
01127         ctl->qnt_ens = -1;
01128         ctl->qnt_m = -1;
01129         ctl->qnt_r = -1;
01130         ctl->qnt_rho = -1;
01131         ctl->qnt_ps = -1;
01132         ctl->qnt_pt = -1;
01133         ctl->qnt_z = -1;
01134         ctl->qnt_p = -1;
01135         ctl->qnt_t = -1;
01136         ctl->qnt_u = -1;
01137         ctl->qnt_v = -1;
01138         ctl->qnt_w = -1;
01139         ctl->qnt_h2o = -1;
01140         ctl->qnt_o3 = -1;
01141         ctl->qnt_theta = -1;
01142         ctl->qnt_vh = -1;
01143         ctl->qnt_vz = -1;
01144         ctl->qnt_pv = -1;
01145         ctl->qnt_tice = -1;
01146         ctl->qnt_tsts = -1;
01147         ctl->qnt_tnat = -1;
01148         ctl->qnt_stat = -1;
01149
01150         /* Read quantities... */
01151         ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
01152         if (ctl->nq > NQ)
01153             ERRMSG("Too many quantities!");
01154         for (int iq = 0; iq < ctl->nq; iq++) {
01155
01156             /* Read quantity name and format... */
01157             scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
01158             scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
01159                 ctl->qnt_format[iq]);
01160
01161             /* Try to identify quantity... */
01162             if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
01163                 ctl->qnt_ens = iq;
01164                 sprintf(ctl->qnt_unit[iq], "-");
01165             } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
01166                 ctl->qnt_m = iq;
01167                 sprintf(ctl->qnt_unit[iq], "kg");
01168             } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
01169                 ctl->qnt_r = iq;
01170                 sprintf(ctl->qnt_unit[iq], "m");
01171             } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
01172                 ctl->qnt_rho = iq;
01173                 sprintf(ctl->qnt_unit[iq], "kg/m^3");
01174             } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
01175                 ctl->qnt_ps = iq;
01176                 sprintf(ctl->qnt_unit[iq], "hPa");
01177             } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
01178                 ctl->qnt_pt = iq;
01179                 sprintf(ctl->qnt_unit[iq], "hPa");
01180             } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
01181                 ctl->qnt_z = iq;
01182                 sprintf(ctl->qnt_unit[iq], "km");
01183             } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
01184                 ctl->qnt_p = iq;
01185                 sprintf(ctl->qnt_unit[iq], "hPa");
01186             } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
01187                 ctl->qnt_t = iq;
01188                 sprintf(ctl->qnt_unit[iq], "K");
01189             } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
01190                 ctl->qnt_u = iq;
01191                 sprintf(ctl->qnt_unit[iq], "m/s");
01192             } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
01193                 ctl->qnt_v = iq;
01194                 sprintf(ctl->qnt_unit[iq], "m/s");
01195             } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
01196                 ctl->qnt_w = iq;
01197                 sprintf(ctl->qnt_unit[iq], "hPa/s");
01198             } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
01199                 ctl->qnt_h2o = iq;
01200                 sprintf(ctl->qnt_unit[iq], "1");
01201             } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
01202                 ctl->qnt_o3 = iq;
01203                 sprintf(ctl->qnt_unit[iq], "1");
01204             } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
01205                 ctl->qnt_theta = iq;

```

```

01206     sprintf(ctl->qnt_unit[iq], "K");
01207 } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
01208     ctl->qnt_vh = iq;
01209     sprintf(ctl->qnt_unit[iq], "m/s");
01210 } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
01211     ctl->qnt_vz = iq;
01212     sprintf(ctl->qnt_unit[iq], "m/s");
01213 } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
01214     ctl->qnt_pv = iq;
01215     sprintf(ctl->qnt_unit[iq], "PVU");
01216 } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
01217     ctl->qnt_tice = iq;
01218     sprintf(ctl->qnt_unit[iq], "K");
01219 } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
01220     ctl->qnt_tsts = iq;
01221     sprintf(ctl->qnt_unit[iq], "K");
01222 } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
01223     ctl->qnt_tnat = iq;
01224     sprintf(ctl->qnt_unit[iq], "K");
01225 } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
01226     ctl->qnt_stat = iq;
01227     sprintf(ctl->qnt_unit[iq], "-");
01228 } else
01229     scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
01230 }
01231
01232 /* Check quantity flags... */
01233 if (ctl->qnt_tsts >= 0)
01234     if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
01235         ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
01236
01237 /* Time steps of simulation... */
01238 ctl->direction =
01239     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
01240 if (ctl->direction != -1 && ctl->direction != 1)
01241     ERRMSG("Set DIRECTION to -1 or 1!");
01242 ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
01243 ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
01244
01245 /* Meteorological data... */
01246 ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
01247 ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
01248 ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
01249 ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
01250 ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
01251 ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
01252 ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
01253 ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
01254 if (ctl->met_np > EP)
01255     ERRMSG("Too many levels!");
01256 for (int ip = 0; ip < ctl->met_np; ip++)
01257     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
01258
01259     = (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "0", NULL);
01260 scan_ctl(filename, argc, argv, "MET_GEOPO", -1, "-", ctl->met_geopot);
01261 scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
01262 ctl->met_dt_out =
01263     scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
01264
01265 /* Isosurface parameters... */
01266 ctl->isosurf
01267     = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
01268 scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
01269
01270 /* Diffusion parameters... */
01271 ctl->turb_dx_trop
01272     = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
01273 ctl->turb_dx_strat
01274     = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
01275 ctl->turb_dz_trop
01276     = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
01277 ctl->turb_dz_strat
01278     = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
01279 ctl->turb_mesox =
01280     scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
01281 ctl->turb_mesoz =
01282     scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
01283
01284 /* Mass and life time... */
01285 ctl->molmass = scan_ctl(filename, argc, argv, "MOLMASS", -1, "1", NULL);
01286 ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
01287 ctl->tdec_strat =
01288     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
01289
01290 /* PSC analysis... */
01291 ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
01292 ctl->psc_hno3 =

```

```

01293     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
01294
01295     /* Output of atmospheric data... */
01296     scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
atm_basename);
01297     scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
01298     ctl->atm_dt_out =
01299         scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
01300     ctl->atm_filter =
01301         (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
01302     ctl->atm_stride =
01303         (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
01304     ctl->atm_type =
01305         (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
01306
01307     /* Output of CSI data... */
01308     scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
csi_basename);
01309     ctl->csi_dt_out =
01310         scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
01311     scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->
csi_obsfile);
01312     ctl->csi_obsmin =
01313         scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
01314     ctl->csi_modmin =
01315         scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
01316     ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
01317     ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
01318     ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
01319     ctl->csi_lon0 =
01320         scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
01321     ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
01322     ctl->csi_nx =
01323         (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
01324     ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
01325     ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
01326     ctl->csi_ny =
01327         (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
01328
01329     /* Output of ensemble data... */
01330     scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
ens_basename);
01331
01332     /* Output of grid data... */
01333     scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
01334         ctl->grid_basename);
01335     scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
grid_gpfile);
01336     ctl->grid_dt_out =
01337         scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
01338     ctl->grid_sparse =
01339         (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
01340     ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
01341     ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
01342     ctl->grid_nz =
01343         (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
01344     ctl->grid_lon0 =
01345         scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
01346     ctl->grid_lon1 =
01347         scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
01348     ctl->grid_nx =
01349         (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
01350     ctl->grid_lat0 =
01351         scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
01352     ctl->grid_lat1 =
01353         scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
01354     ctl->grid_ny =
01355         (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
01356
01357     /* Output of profile data... */
01358     scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
01359         ctl->prof_basename);
01360     scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
prof_obsfile);
01361     ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
01362     ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
01363     ctl->prof_nz =
01364         (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
01365     ctl->prof_lon0 =
01366         scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
01367     ctl->prof_lon1 =
01368         scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
01369     ctl->prof_nx =
01370         (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
01371     ctl->prof_lat0 =
01372         scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
01373     ctl->prof_lat1 =

```

```

01374     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
01375     ctl->prof_ny =
01376         (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
01377
01378     /* Output of station data... */
01379     scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
01380             ctl->stat_basename);
01381     ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
01382     ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
01383     ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
01384 }

```

Here is the call graph for this function:



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

Read meteorological data file.

Definition at line 1388 of file libtrac.c.

```

01391     {
01392
01393     char cmd[2 * LEN], levname[LEN], tstr[10];
01394
01395     float help[EX * EY];
01396
01397     int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
01398
01399     size_t np, nx, ny;
01400
01401     /* Write info... */
01402     printf("Read meteorological data: %s\n", filename);
01403
01404     /* Get time from filename... */
01405     sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
01406     year = atoi(tstr);
01407     sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
01408     mon = atoi(tstr);
01409     sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
01410     day = atoi(tstr);
01411     sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
01412     hour = atoi(tstr);
01413     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
01414
01415     /* Open netCDF file... */
01416     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
01417
01418         /* Try to stage meteo file... */
01419         if (ctl->met_stage[0] != '-') {
01420             sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
01421                     year, mon, day, hour, filename);
01422             if (system(cmd) != 0)
01423                 ERRMSG("Error while staging meteo data!");
01424         }
01425
01426         /* Try to open again... */
01427         if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
01428             WARN("File not found!");
01429             return 0;
01430         }
01431     }
01432 }

```

```

01433  /* Get dimensions... */
01434  NC(nc_inq_dimid(ncid, "lon", &dimid));
01435  NC(nc_inq_dimlen(ncid, dimid, &nx));
01436  if (nx < 2 || nx > EX)
01437      ERRMSG("Number of longitudes out of range!");
01438
01439  NC(nc_inq_dimid(ncid, "lat", &dimid));
01440  NC(nc_inq_dimlen(ncid, dimid, &ny));
01441  if (ny < 2 || ny > EY)
01442      ERRMSG("Number of latitudes out of range!");
01443
01444  sprintf(levname, "lev");
01445  NC(nc_inq_dimid(ncid, levname, &dimid));
01446  NC(nc_inq_dimlen(ncid, dimid, &np));
01447  if (np == 1) {
01448      sprintf(levname, "lev_2");
01449      NC(nc_inq_dimid(ncid, levname, &dimid));
01450      NC(nc_inq_dimlen(ncid, dimid, &np));
01451  }
01452  if (np < 2 || np > EP)
01453      ERRMSG("Number of levels out of range!");
01454
01455  /* Store dimensions... */
01456  met->np = (int) np;
01457  met->nx = (int) nx;
01458  met->ny = (int) ny;
01459
01460  /* Get horizontal grid... */
01461  NC(nc_inq_varid(ncid, "lon", &varid));
01462  NC(nc_get_var_double(ncid, varid, met->lon));
01463  NC(nc_inq_varid(ncid, "lat", &varid));
01464  NC(nc_get_var_double(ncid, varid, met->lat));
01465
01466  /* Read meteorological data... */
01467  read_met_help(ncid, "t", "T", met, met->t, 1.0);
01468  read_met_help(ncid, "u", "U", met, met->u, 1.0);
01469  read_met_help(ncid, "v", "V", met, met->v, 1.0);
01470  read_met_help(ncid, "w", "W", met, met->w, 0.01f);
01471  read_met_help(ncid, "q", "Q", met, met->h2o, (float) (MA / 18.01528));
01472  read_met_help(ncid, "o3", "O3", met, met->o3, (float) (MA / 48.00));
01473
01474  /* Meteo data on pressure levels... */
01475  if (ctl->met_np <= 0) {
01476
01477      /* Read pressure levels from file... */
01478      NC(nc_inq_varid(ncid, levname, &varid));
01479      NC(nc_get_var_double(ncid, varid, met->p));
01480      for (ip = 0; ip < met->np; ip++)
01481          met->p[ip] /= 100.;
01482
01483      /* Extrapolate data for lower boundary... */
01484      read_met_extrapolate(met);
01485  }
01486
01487  /* Meteo data on model levels... */
01488  else {
01489
01490      /* Read pressure data from file... */
01491      read_met_help(ncid, "pl", "PL", met, met->p1, 0.01f);
01492
01493      /* Interpolate from model levels to pressure levels... */
01494      read_met_ml2pl(ctl, met, met->t);
01495      read_met_ml2pl(ctl, met, met->u);
01496      read_met_ml2pl(ctl, met, met->v);
01497      read_met_ml2pl(ctl, met, met->w);
01498      read_met_ml2pl(ctl, met, met->h2o);
01499      read_met_ml2pl(ctl, met, met->o3);
01500
01501      /* Set pressure levels... */
01502      met->np = ctl->met_np;
01503      for (ip = 0; ip < met->np; ip++)
01504          met->p[ip] = ctl->met_p[ip];
01505  }
01506
01507  /* Check ordering of pressure levels... */
01508  for (ip = 1; ip < met->np; ip++)
01509      if (met->p[ip - 1] < met->p[ip])
01510          ERRMSG("Pressure levels must be descending!");
01511
01512  /* Read surface pressure... */
01513  if (nc_inq_varid(ncid, "ps", &varid) == NC_NOERR
01514      || nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
01515      NC(nc_get_var_float(ncid, varid, help));
01516      for (iy = 0; iy < met->ny; iy++)
01517          for (ix = 0; ix < met->nx; ix++)
01518              met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
01519  } else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR

```

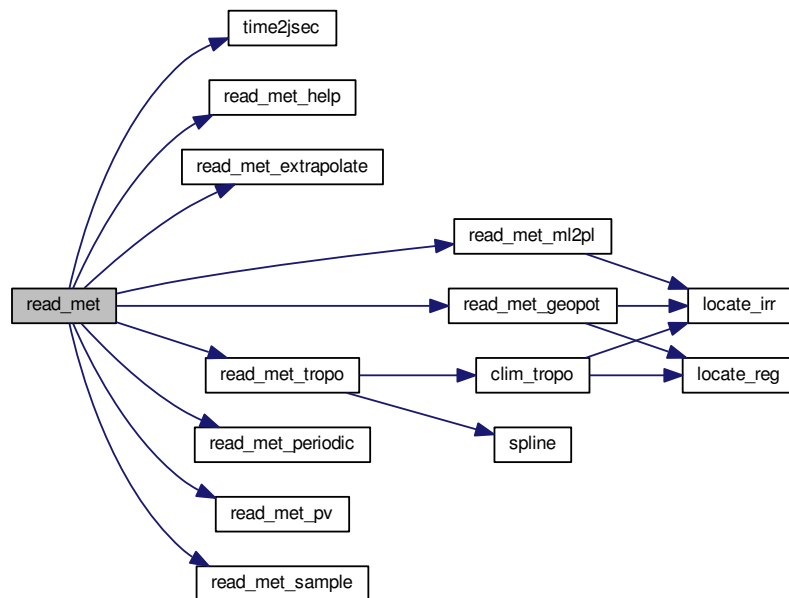


```

01520         || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
01521     NC(nc_get_var_float(ncid, varid, help));
01522     for (iy = 0; iy < met->ny; iy++)
01523         for (ix = 0; ix < met->nx; ix++)
01524             met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
01525 } else
01526     for (ix = 0; ix < met->nx; ix++)
01527         for (iy = 0; iy < met->ny; iy++)
01528             met->ps[ix][iy] = met->p[0];
01529
01530 /* Create periodic boundary conditions... */
01531 read_met_periodic(met);
01532
01533 /* Calculate geopotential heights... */
01534 read_met_geopot(ctl, met);
01535
01536 /* Calculate potential vorticity... */
01537 read_met_pv(met);
01538
01539 /* Calculate tropopause pressure... */
01540 read_met_tropo(ctl, met);
01541
01542 /* Downsampling... */
01543 read_met_sample(ctl, met);
01544
01545 /* Close file... */
01546 NC(nc_close(ncid));
01547
01548 /* Return success... */
01549 return 1;
01550 }

```

Here is the call graph for this function:



5.19.2.20 void read_met_extrapolate (met_t * met)

Extrapolate meteorological data at lower boundary.

Definition at line 1554 of file [libtrac.c](#).

```

01555         {
01556
01557     int ip, ip0, ix, iy;
01558
01559     /* Loop over columns... */
01560 #pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
01561     for (ix = 0; ix < met->nx; ix++)
01562         for (iy = 0; iy < met->ny; iy++) {
01563
01564             /* Find lowest valid data point... */
01565             for (ip0 = met->np - 1; ip0 >= 0; ip0--)
01566                 if (!gsl_finite(met->t[ix][iy][ip0])
01567                     || !gsl_finite(met->u[ix][iy][ip0])
01568                     || !gsl_finite(met->v[ix][iy][ip0])
01569                     || !gsl_finite(met->w[ix][iy][ip0]))
01570                 break;
01571
01572             /* Extrapolate... */
01573             for (ip = ip0; ip >= 0; ip--) {
01574                 met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
01575                 met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
01576                 met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
01577                 met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
01578                 met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
01579                 met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
01580             }
01581         }
01582 }

```

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

Calculate geopotential heights.

Definition at line 1586 of file libtrac.c.

```

01588         {
01589
01590     const int dx = 6, dy = 4;
01591
01592     static double logp[EP], topo_lat[EY], topo_lon[EX], topo_z[EX][EY];
01593
01594     static float help[EX][EY][EP];
01595
01596     static int init, topo_nx = -1, topo_ny;
01597
01598     FILE *in;
01599
01600     char line[LEN];
01601
01602     double lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;
01603
01604     int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;
01605
01606     /* Initialize geopotential heights... */
01607 #pragma omp parallel for default(shared) private(ix,iy,ip)
01608     for (ix = 0; ix < met->nx; ix++)
01609         for (iy = 0; iy < met->ny; iy++)
01610             for (ip = 0; ip < met->np; ip++)
01611                 met->z[ix][iy][ip] = GSL_NAN;
01612
01613     /* Check filename... */
01614     if (ctl->met_geopot[0] == '-')
01615         return;
01616
01617     /* Read surface geopotential... */
01618     if (!init) {
01619         init = 1;
01620
01621         /* Write info... */
01622         printf("Read surface geopotential: %s\n", ctl->met_geopot);
01623
01624         /* Open file... */
01625         if (!(in = fopen(ctl->met_geopot, "r")))
01626             ERRMSG("Cannot open file!");
01627
01628         /* Read data... */
01629         while (fgets(line, LEN, in))
01630             if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rz) == 3) {
01631                 if (rlon != rlon_old) {
01632                     if (++topo_nx > EX)

```

```

01633         ERRMSG("Too many longitudes!");
01634         topo_ny = 0;
01635     }
01636     rlon_old = rlon;
01637     topo_lon[topo_nx] = rlon;
01638     topo_lat[topo_ny] = rlat;
01639     topo_z[topo_nx][topo_ny] = rz;
01640     if ((++topo_ny) > EY)
01641         ERRMSG("Too many latitudes!");
01642 }
01643 if ((++topo_nx) > EX)
01644     ERRMSG("Too many longitudes!");
01645
01646 /* Close file... */
01647 fclose(in);
01648
01649 /* Check grid spacing... */
01650 if (fabs(met->lon[0] - met->lon[1]) != fabs(topo_lon[0] - topo_lon[1])
01651     || fabs(met->lat[0] - met->lat[1]) != fabs(topo_lat[0] - topo_lat[1]))
01652     WARN("Grid spacing does not match!");
01653
01654 /* Calculate log pressure... */
01655 for (ip = 0; ip < met->np; ip++)
01656     logp[ip] = log(met->p[ip]);
01657 }
01658
01659 /* Apply hydrostatic equation to calculate geopotential heights... */
01660 #pragma omp parallel for default(shared) private(ix,iy,lon,lat,tx,ty,z0,z1,ip0,ts,ip)
01661 for (ix = 0; ix < met->nx; ix++) {
01662
01663     /* Get longitude index... */
01664     lon = met->lon[ix];
01665     if (lon < topo_lon[0])
01666         lon += 360;
01667     else if (lon > topo_lon[topo_nx - 1])
01668         lon -= 360;
01669     tx = locate_reg(topo_lon, topo_nx, lon);
01670
01671     /* Loop over latitudes... */
01672     for (iy = 0; iy < met->ny; iy++) {
01673
01674         /* Get latitude index... */
01675         lat = met->lat[iy];
01676         ty = locate_reg(topo_lat, topo_ny, lat);
01677
01678         /* Get surface height... */
01679         z0 = LIN(topo_lon[tx], topo_z[tx][ty],
01680                 topo_lon[tx + 1], topo_z[tx + 1][ty], lon);
01681         z1 = LIN(topo_lon[tx], topo_z[tx][ty + 1],
01682                 topo_lon[tx + 1], topo_z[tx + 1][ty + 1], lon);
01683         z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);
01684
01685         /* Find surface pressure level... */
01686         ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
01687
01688         /* Get surface data... */
01689         ts =
01690             LIN(met->p[ip0],
01691                 TVIRT(met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]),
01692                 met->p[ip0 + 1],
01693                 TVIRT(met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]),
01694                 met->ps[ix][iy]);
01695
01696         /* Upper part of profile... */
01697         met->z[ix][iy][ip0 + 1]
01698             = (float) (z0 + RI / MA / G0 * 0.5
01699                     * (ts + TVIRT(met->t[ix][iy][ip0 + 1],
01700                                   met->h2o[ix][iy][ip0 + 1]))
01701                     * (log(met->ps[ix][iy]) - logp[ip0 + 1]));
01702         for (ip = ip0 + 2; ip < met->np; ip++)
01703             met->z[ix][iy][ip]
01704                 = (float) (met->z[ix][iy][ip - 1] + RI / MA / G0 * 0.5 *
01705                           (TVIRT(met->t[ix][iy][ip - 1], met->h2o[ix][iy][ip - 1])
01706                             + TVIRT(met->t[ix][iy][ip], met->h2o[ix][iy][ip]))
01707                           * (logp[ip - 1] - logp[ip]));
01708     }
01709 }
01710
01711 /* Smoothing... */
01712 #pragma omp parallel for default(shared) private(ix,iy,ip,n,ix2,ix3,iy2)
01713 for (ix = 0; ix < met->nx; ix++)
01714     for (iy = 0; iy < met->ny; iy++)
01715         for (ip = 0; ip < met->np; ip++) {
01716             n = 0;
01717             help[ix][iy][ip] = 0;
01718             for (ix2 = ix - dx; ix2 <= ix + dx; ix2++) {
01719                 ix3 = ix2;

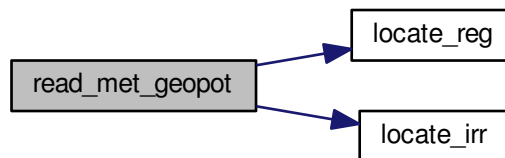
```

```

01720         if (ix3 < 0)
01721             ix3 += met->nx;
01722         else if (ix3 >= met->nx)
01723             ix3 -= met->nx;
01724         for (iy2 = GSL_MAX(iy - dy, 0);
01725              iy2 <= GSL_MIN(iy + dy, met->ny - 1); iy2++)
01726             if (gsl_finite(met->z[ix3][iy2][ip])) {
01727                 help[ix][iy][ip] += met->z[ix3][iy2][ip];
01728                 n++;
01729             }
01730     }
01731     if (n > 0)
01732         help[ix][iy][ip] /= (float) n;
01733     else
01734         help[ix][iy][ip] = GSL_NAN;
01735 }
01736
01737 /* Copy data... */
01738 #pragma omp parallel for default(shared) private(ix,iy,ip)
01739 for (ix = 0; ix < met->nx; ix++)
01740     for (iy = 0; iy < met->ny; iy++)
01741         for (ip = 0; ip < met->np; ip++)
01742             met->z[ix][iy][ip] = help[ix][iy][ip];
01743 }

```

Here is the call graph for this function:



5.19.2.22 void read_met_help (int ncid, char * varname, char * varname2, met_t * met, float dest[EX][EY][EP], float scl)

Read and convert variable from meteorological data file.

Definition at line 1747 of file libtrac.c.

```

01753     {
01754
01755         float *help;
01756
01757         int ip, ix, iy, varid;
01758
01759         /* Check if variable exists... */
01760         if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
01761             if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
01762                 return;
01763
01764         /* Allocate... */
01765         ALLOC(help, float, EX * EY * EP);
01766
01767         /* Read data... */
01768         NC(nc_get_var_float(ncid, varid, help));
01769
01770         /* Copy and check data... */
01771         #pragma omp parallel for default(shared) private(ix,iy,ip)
01772         for (ix = 0; ix < met->nx; ix++)
01773             for (iy = 0; iy < met->ny; iy++)
01774                 for (ip = 0; ip < met->np; ip++) {
01775                     dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
01776                     if (fabsf(dest[ix][iy][ip]) < 1e14f)
01777                         dest[ix][iy][ip] *= scl;
01778                 }
01779     }

```

```

01778         else
01779             dest[ix][iy][ip] = GSL_NAN;
01780     }
01781
01782     /* Free... */
01783     free(help);
01784 }

```

5.19.2.23 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 1788 of file libtrac.c.

```

01791     {
01792
01793     double aux[EP], p[EP], pt;
01794
01795     int ip, ip2, ix, iy;
01796
01797     /* Loop over columns... */
01798     #pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
01799     for (ix = 0; ix < met->nx; ix++)
01800         for (iy = 0; iy < met->ny; iy++) {
01801
01802             /* Copy pressure profile... */
01803             for (ip = 0; ip < met->np; ip++)
01804                 p[ip] = met->p[ix][iy][ip];
01805
01806             /* Interpolate... */
01807             for (ip = 0; ip < ctl->met_np; ip++) {
01808                 pt = ctl->met_p[ip];
01809                 if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
01810                     pt = p[0];
01811                 else if ((pt > p[met->np - 1] && p[1] > p[0])
01812                        || (pt < p[met->np - 1] && p[1] < p[0]))
01813                     pt = p[met->np - 1];
01814                 ip2 = locate_irr(p, met->np, pt);
01815                 aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
01816                             p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
01817             }
01818
01819             /* Copy data... */
01820             for (ip = 0; ip < ctl->met_np; ip++)
01821                 var[ix][iy][ip] = (float) aux[ip];
01822         }
01823     }

```

Here is the call graph for this function:



5.19.2.24 void read_met_periodic (met_t * met)

Create meteorological data with periodic boundary conditions.

Definition at line 1827 of file libtrac.c.

```

01828         {
01829
01830         /* Check longitudes... */
01831         if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
01832                 + met->lon[1] - met->lon[0] - 360) < 0.01))
01833             return;
01834
01835         /* Increase longitude counter... */
01836         if ((++met->nx) > EX)
01837             ERRMSG("Cannot create periodic boundary conditions!");
01838
01839         /* Set longitude... */
01840         met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
lon[0];
01841
01842         /* Loop over latitudes and pressure levels... */
01843         #pragma omp parallel for default(shared)
01844         for (int iy = 0; iy < met->ny; iy++) {
01845             met->ps[met->nx - 1][iy] = met->ps[0][iy];
01846             met->pt[met->nx - 1][iy] = met->pt[0][iy];
01847             for (int ip = 0; ip < met->np; ip++) {
01848                 met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
01849                 met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
01850                 met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
01851                 met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
01852                 met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
01853                 met->pv[met->nx - 1][iy][ip] = met->pv[0][iy][ip];
01854                 met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
01855                 met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01856             }
01857         }
01858     }

```

5.19.2.25 void read_met_pv (met_t * met)

Calculate potential vorticity.

Definition at line 1862 of file libtrac.c.

```

01863         {
01864
01865         double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
01866                 dtdp, dudp, dvdp, latr, vort, pows[EP];
01867
01868         int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
01869
01870         /* Set powers... */
01871         for (ip = 0; ip < met->np; ip++)
01872             pows[ip] = pow(1000. / met->p[ip], 0.286);
01873
01874         /* Loop over grid points... */
01875         #pragma omp parallel for default(shared)
01876         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)
01877         for (ix = 0; ix < met->nx; ix++) {
01878
01879             /* Set indices... */
01880             ix0 = GSL_MAX(ix - 1, 0);
01881             ix1 = GSL_MIN(ix + 1, met->nx - 1);
01882
01883             /* Loop over grid points... */
01884             for (iy = 0; iy < met->ny; iy++) {
01885
01886                 /* Set indices... */
01887                 iy0 = GSL_MAX(iy - 1, 0);
01888                 iy1 = GSL_MIN(iy + 1, met->ny - 1);
01889
01890                 /* Set auxiliary variables... */
01891                 latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
01892                 dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
01893                 dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
01894                 c0 = cos(met->lat[iy0] / 180. * M_PI);
01895                 c1 = cos(met->lat[iy1] / 180. * M_PI);
01896                 cr = cos(latr / 180. * M_PI);
01897                 vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
01898
01899                 /* Loop over grid points... */
01900                 for (ip = 0; ip < met->np; ip++) {
01901
01902                     /* Get gradients in longitude... */
01903                     dtdx = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;

```

```

01903     dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
01904
01905     /* Get gradients in latitude... */
01906     dtdy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
01907     dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
01908
01909     /* Set indices... */
01910     ip0 = GSL_MAX(ip - 1, 0);
01911     ip1 = GSL_MIN(ip + 1, met->np - 1);
01912
01913     /* Get gradients in pressure... */
01914     dp0 = 100. * (met->p[ip] - met->p[ip0]);
01915     dp1 = 100. * (met->p[ip1] - met->p[ip]);
01916     if (ip != ip0 && ip != ip1) {
01917         denom = dp0 * dp1 * (dp0 + dp1);
01918         dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
01919                 - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
01920                 + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
01921                 / denom;
01922         dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
01923                 - dp1 * dp1 * met->u[ix][iy][ip0]
01924                 + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
01925                 / denom;
01926         dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
01927                 - dp1 * dp1 * met->v[ix][iy][ip0]
01928                 + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
01929                 / denom;
01930     } else {
01931         denom = dp0 + dp1;
01932         dtdp =
01933             (met->t[ix][iy][ip1] * pows[ip1] -
01934              met->t[ix][iy][ip0] * pows[ip0]) / denom;
01935         dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
01936         dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
01937     }
01938
01939     /* Calculate PV... */
01940     met->pv[ix][iy][ip] = (float)
01941         (1e6 * G0 *
01942          (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
01943 }
01944 }
01945 }
01946
01947 /* Fix for polar regions... */
01948 #pragma omp parallel for default(shared) private(ix,ip)
01949 for (ix = 0; ix < met->nx; ix++)
01950     for (ip = 0; ip < met->np; ip++) {
01951         met->pv[ix][0][ip]
01952             = met->pv[ix][1][ip]
01953             = met->pv[ix][2][ip];
01954         met->pv[ix][met->ny - 1][ip]
01955             = met->pv[ix][met->ny - 2][ip]
01956             = met->pv[ix][met->ny - 3][ip];
01957     }
01958 }

```

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

Downsampling of meteorological data.

Definition at line 1962 of file libtrac.c.

```

01964     {
01965
01966         met_t *help;
01967
01968         float w, wsum;
01969
01970         int ip, ip2, ix, ix2, ix3, iy, iy2;
01971
01972         /* Check parameters... */
01973         if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
01974             && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
01975             return;
01976
01977         /* Allocate... */
01978         ALLOC(help, met_t, 1);
01979
01980         /* Copy data... */

```

```

01981 help->nx = met->nx;
01982 help->ny = met->ny;
01983 help->np = met->np;
01984 memcpy(help->lon, met->lon, sizeof(met->lon));
01985 memcpy(help->lat, met->lat, sizeof(met->lat));
01986 memcpy(help->p, met->p, sizeof(met->p));
01987
01988 /* Smoothing... */
01989 for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
01990     for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
01991         for (ip = 0; ip < met->np; ip += ctl->met_dp) {
01992             help->ps[ix][iy] = 0;
01993             help->pt[ix][iy] = 0;
01994             help->z[ix][iy][ip] = 0;
01995             help->t[ix][iy][ip] = 0;
01996             help->u[ix][iy][ip] = 0;
01997             help->v[ix][iy][ip] = 0;
01998             help->w[ix][iy][ip] = 0;
01999             help->pv[ix][iy][ip] = 0;
02000             help->h2o[ix][iy][ip] = 0;
02001             help->o3[ix][iy][ip] = 0;
02002             wsum = 0;
02003             for (ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1; ix2++) {
02004                 ix3 = ix2;
02005                 if (ix3 < 0)
02006                     ix3 += met->nx;
02007                 else if (ix3 >= met->nx)
02008                     ix3 -= met->nx;
02009
02010                 for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
02011                     iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
02012                     for (ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
02013                         ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
02014                         w = (float) (1.0 - fabs(ix - ix2) / ctl->met_sx)
02015                             * (float) (1.0 - fabs(iy - iy2) / ctl->met_sy)
02016                             * (float) (1.0 - fabs(ip - ip2) / ctl->met_sp);
02017                         help->ps[ix][iy] += w * met->ps[ix3][iy2];
02018                         help->pt[ix][iy] += w * met->pt[ix3][iy2];
02019                         help->z[ix][iy][ip] += w * met->z[ix3][iy2][ip2];
02020                         help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
02021                         help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
02022                         help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
02023                         help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
02024                         help->pv[ix][iy][ip] += w * met->pv[ix3][iy2][ip2];
02025                         help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
02026                         help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
02027                         wsum += w;
02028                     }
02029             }
02030             help->ps[ix][iy] /= wsum;
02031             help->pt[ix][iy] /= wsum;
02032             help->t[ix][iy][ip] /= wsum;
02033             help->z[ix][iy][ip] /= wsum;
02034             help->u[ix][iy][ip] /= wsum;
02035             help->v[ix][iy][ip] /= wsum;
02036             help->w[ix][iy][ip] /= wsum;
02037             help->pv[ix][iy][ip] /= wsum;
02038             help->h2o[ix][iy][ip] /= wsum;
02039             help->o3[ix][iy][ip] /= wsum;
02040         }
02041     }
02042 }
02043
02044 /* Downsampling... */
02045 met->nx = 0;
02046 for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
02047     met->lon[met->nx] = help->lon[ix];
02048     met->ny = 0;
02049     for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
02050         met->lat[met->ny] = help->lat[iy];
02051         met->ps[met->nx][met->ny] = help->ps[ix][iy];
02052         met->pt[met->nx][met->ny] = help->pt[ix][iy];
02053         met->np = 0;
02054         for (ip = 0; ip < help->np; ip += ctl->met_dp) {
02055             met->p[met->np] = help->p[ip];
02056             met->z[met->nx][met->ny][met->np] = help->z[ix][iy][ip];
02057             met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
02058             met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
02059             met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
02060             met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
02061             met->pv[met->nx][met->ny][met->np] = help->pv[ix][iy][ip];
02062             met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
02063             met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
02064             met->np++;
02065         }
02066         met->ny++;
02067     }

```



```

02068     met->nx++;
02069 }
02070
02071 /* Free... */
02072 free(help);
02073 }

```

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

Calculate tropopause pressure.

Definition at line 2077 of file libtrac.c.

```

02079     {
02080
02081     double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
02082           th2[200], z[EP], z2[200];
02083
02084     int found, ix, iy, iz, iz2;
02085
02086     /* Get altitude and pressure profiles... */
02087     for (iz = 0; iz < met->np; iz++)
02088         z[iz] = Z(met->p[iz]);
02089     for (iz = 0; iz <= 170; iz++) {
02090         z2[iz] = 4.5 + 0.1 * iz;
02091         p2[iz] = P(z2[iz]);
02092     }
02093
02094     /* Do not calculate tropopause... */
02095     if (ctl->met_tropo == 0)
02096         for (ix = 0; ix < met->nx; ix++)
02097             for (iy = 0; iy < met->ny; iy++)
02098                 met->pt[ix][iy] = GSL_NAN;
02099
02100     /* Use tropopause climatology... */
02101     else if (ctl->met_tropo == 1) {
02102 #pragma omp parallel for default(shared) private(ix,iy)
02103         for (ix = 0; ix < met->nx; ix++)
02104             for (iy = 0; iy < met->ny; iy++)
02105                 met->pt[ix][iy] = clim_tropo(met->time, met->lat[iy]);
02106     }
02107
02108     /* Use cold point... */
02109     else if (ctl->met_tropo == 2) {
02110
02111         /* Loop over grid points... */
02112 #pragma omp parallel for default(shared) private(ix,iy,iz,t,t2)
02113         for (ix = 0; ix < met->nx; ix++)
02114             for (iy = 0; iy < met->ny; iy++) {
02115
02116                 /* Interpolate temperature profile... */
02117                 for (iz = 0; iz < met->np; iz++)
02118                     t[iz] = met->t[ix][iy][iz];
02119                 spline(z, t, met->np, z2, t2, 171);
02120
02121                 /* Find minimum... */
02122                 iz = (int) gsl_stats_min_index(t2, 1, 171);
02123                 if (iz <= 0 || iz >= 170)
02124                     met->pt[ix][iy] = GSL_NAN;
02125                 else
02126                     met->pt[ix][iy] = p2[iz];
02127             }
02128     }
02129
02130     /* Use WMO definition... */
02131     else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
02132
02133         /* Loop over grid points... */
02134 #pragma omp parallel for default(shared) private(ix,iy,iz,iz2,t,t2,found)
02135         for (ix = 0; ix < met->nx; ix++)
02136             for (iy = 0; iy < met->ny; iy++) {
02137
02138                 /* Interpolate temperature profile... */
02139                 for (iz = 0; iz < met->np; iz++)
02140                     t[iz] = met->t[ix][iy][iz];
02141                 spline(z, t, met->np, z2, t2, 161);
02142
02143                 /* Find 1st tropopause... */
02144                 met->pt[ix][iy] = GSL_NAN;
02145                 for (iz = 0; iz <= 140; iz++) {

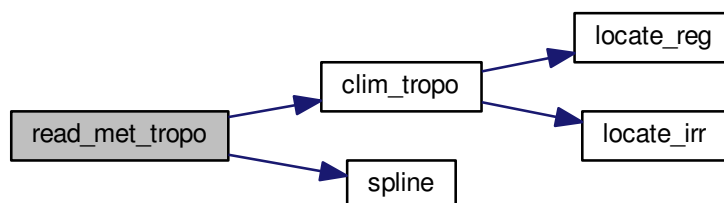
```

```

02146         found = 1;
02147         for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
02148             if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
02149                 * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
02150                 found = 0;
02151                 break;
02152             }
02153         if (found) {
02154             if (iz > 0 && iz < 140)
02155                 met->pt[ix][iy] = p2[iz];
02156             break;
02157         }
02158     }
02159 }
02160
02161 /* Find 2nd tropopause... */
02162 if (ctl->met_tropo == 4) {
02163     met->pt[ix][iy] = GSL_NAN;
02164     for (; iz <= 140; iz++) {
02165         found = 1;
02166         for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
02167             if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
02168                 * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) < 3.0) {
02169                 found = 0;
02170                 break;
02171             }
02172         if (found)
02173             break;
02174     }
02175     for (; iz <= 140; iz++) {
02176         found = 1;
02177         for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
02178             if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
02179                 * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
02180                 found = 0;
02181                 break;
02182             }
02183         if (found) {
02184             if (iz > 0 && iz < 140)
02185                 met->pt[ix][iy] = p2[iz];
02186             break;
02187         }
02188     }
02189 }
02190 }
02191
02192 /* Use dynamical tropopause... */
02193 else if (ctl->met_tropo == 5) {
02194
02195     /* Loop over grid points... */
02196     #pragma omp parallel for default(shared) private(ix,iy,iz,pv,pv2,th,th2)
02197     for (ix = 0; ix < met->nx; ix++)
02198         for (iy = 0; iy < met->ny; iy++) {
02199
02200             /* Interpolate potential vorticity profile... */
02201             for (iz = 0; iz < met->np; iz++)
02202                 pv[iz] = met->pv[ix][iy][iz];
02203             spline(z, pv, met->np, z2, pv2, 161);
02204
02205             /* Interpolate potential temperature profile... */
02206             for (iz = 0; iz < met->np; iz++)
02207                 th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
02208             spline(z, th, met->np, z2, th2, 161);
02209
02210             /* Find dynamical tropopause 3.5 PVU + 380 K */
02211             met->pt[ix][iy] = GSL_NAN;
02212             for (iz = 0; iz <= 160; iz++)
02213                 if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
02214                     if (iz > 0 && iz < 160)
02215                         met->pt[ix][iy] = p2[iz];
02216                     break;
02217                 }
02218         }
02219     }
02220
02221     else
02222         ERRMSG("Cannot calculate tropopause!");
02223 }

```

Here is the call graph for this function:



5.19.2.28 `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 2227 of file `libtrac.c`.

```

02234         {
02235
02236     FILE *in = NULL;
02237
02238     char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
02239         msg[2 * LEN], rvarname[LEN], rval[LEN];
02240
02241     int contain = 0, i;
02242
02243     /* Open file... */
02244     if (filename[strlen(filename) - 1] != '-')
02245         if (!(in = fopen(filename, "r")))
02246             ERRMSG("Cannot open file!");
02247
02248     /* Set full variable name... */
02249     if (arridx >= 0) {
02250         sprintf(fullname1, "%s[%d]", varname, arridx);
02251         sprintf(fullname2, "%s[*]", varname);
02252     } else {
02253         sprintf(fullname1, "%s", varname);
02254         sprintf(fullname2, "%s", varname);
02255     }
02256
02257     /* Read data... */
02258     if (in != NULL)
02259         while (fgets(line, LEN, in))
02260             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
02261                 if (strcasecmp(rvarname, fullname1) == 0 ||
02262                     strcasecmp(rvarname, fullname2) == 0) {
02263                     contain = 1;
02264                     break;
02265                 }
02266     for (i = 1; i < argc - 1; i++)
02267         if (strcasecmp(argv[i], fullname1) == 0 ||
02268             strcasecmp(argv[i], fullname2) == 0) {
02269             sprintf(rval, "%s", argv[i + 1]);
02270             contain = 1;
02271             break;
02272         }
02273
02274     /* Close file... */
02275     if (in != NULL)
02276         fclose(in);
02277
02278     /* Check for missing variables... */
02279     if (!contain) {
02280         if (strlen(defvalue) > 0)
02281             sprintf(rval, "%s", defvalue);
  
```

```

02282     else {
02283         sprintf(msg, "Missing variable %s!\n", fullname1);
02284         ERRMSG(msg);
02285     }
02286 }
02287
02288 /* Write info... */
02289 printf("%s = %s\n", fullname1, rval);
02290
02291 /* Return values... */
02292 if (value != NULL)
02293     sprintf(value, "%s", rval);
02294 return atof(rval);
02295 }

```

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

Spline interpolation.

Definition at line 2299 of file libtrac.c.

```

02305     {
02306
02307     gsl_interp_accel *acc;
02308
02309     gsl_spline *s;
02310
02311     /* Allocate... */
02312     acc = gsl_interp_accel_alloc();
02313     s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
02314
02315     /* Interpolate temperature profile... */
02316     gsl_spline_init(s, x, y, (size_t) n);
02317     for (int i = 0; i < n2; i++)
02318         y2[i] = gsl_spline_eval(s, x2[i], acc);
02319
02320     /* Free... */
02321     gsl_spline_free(s);
02322     gsl_interp_accel_free(acc);
02323 }

```

5.19.2.30 double stddev (double * data, int n)

Calculate standard deviation.

Definition at line 2327 of file libtrac.c.

```

02329     {
02330
02331     if (n <= 0)
02332         return 0;
02333
02334     double avg = 0, rms = 0;
02335
02336     for (int i = 0; i < n; ++i)
02337         avg += data[i];
02338     avg /= n;
02339
02340     for (int i = 0; i < n; ++i)
02341         rms += SQR(data[i] - avg);
02342
02343     return sqrt(rms / (n - 1));
02344 }

```

5.19.2.31 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 2348 of file libtrac.c.

```

02356         {
02357
02358     struct tm t0, t1;
02359
02360     t0.tm_year = 100;
02361     t0.tm_mon = 0;
02362     t0.tm_mday = 1;
02363     t0.tm_hour = 0;
02364     t0.tm_min = 0;
02365     t0.tm_sec = 0;
02366
02367     t1.tm_year = year - 1900;
02368     t1.tm_mon = mon - 1;
02369     t1.tm_mday = day;
02370     t1.tm_hour = hour;
02371     t1.tm_min = min;
02372     t1.tm_sec = sec;
02373
02374     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
02375 }
```

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

Measure wall-clock time.

Definition at line 2379 of file libtrac.c.

```

02382         {
02383
02384     static double starttime[NTIMER], runtime[NTIMER];
02385
02386     /* Check id... */
02387     if (id < 0 || id >= NTIMER)
02388         ERRMSG("Too many timers!");
02389
02390     /* Start timer... */
02391     if (mode == 1) {
02392         if (starttime[id] <= 0)
02393             starttime[id] = omp_get_wtime();
02394         else
02395             ERRMSG("Timer already started!");
02396     }
02397
02398     /* Stop timer... */
02399     else if (mode == 2) {
02400         if (starttime[id] > 0) {
02401             runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
02402             starttime[id] = -1;
02403         }
02404     }
02405
02406     /* Print timer... */
02407     else if (mode == 3) {
02408         printf("%s = %.3f s\n", name, runtime[id]);
02409         runtime[id] = 0;
02410     }
02411 }
```

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

Write atmospheric data.

Definition at line 2415 of file libtrac.c.

```

02419     {
02420
02421     FILE *in, *out;
02422
02423     char line[LEN];
02424
02425     double r, t0, t1;
02426
02427     int ip, iq, year, mon, day, hour, min, sec;
02428
02429     /* Set time interval for output... */
02430     t0 = t - 0.5 * ctl->dt_mod;
02431     t1 = t + 0.5 * ctl->dt_mod;
02432
02433     /* Write info... */
02434     printf("Write atmospheric data: %s\n", filename);
02435
02436     /* Write ASCII data... */
02437     if (ctl->atm_type == 0) {
02438
02439         /* Check if gnuplot output is requested... */
02440         if (ctl->atm_gpfile[0] != '-') {
02441
02442             /* Create gnuplot pipe... */
02443             if (!(out = popen("gnuplot", "w")))
02444                 ERRMSG("Cannot create pipe to gnuplot!");
02445
02446             /* Set plot filename... */
02447             fprintf(out, "set out \"%s.png\"\n", filename);
02448
02449             /* Set time string... */
02450             jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02451             fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02452                     year, mon, day, hour, min);
02453
02454             /* Dump gnuplot file to pipe... */
02455             if (!(in = fopen(ctl->atm_gpfile, "r")))
02456                 ERRMSG("Cannot open file!");
02457             while (fgets(line, LEN, in))
02458                 fprintf(out, "%s", line);
02459             fclose(in);
02460         }
02461     }
02462     else {
02463
02464         /* Create file... */
02465         if (!(out = fopen(filename, "w")))
02466             ERRMSG("Cannot create file!");
02467     }
02468
02469     /* Write header... */
02470     fprintf(out,
02471            "# $1 = time [s]\n"
02472            "# $2 = altitude [km]\n"
02473            "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02474     for (iq = 0; iq < ctl->nq; iq++)
02475         fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
02476                 ctl->qnt_unit[iq]);
02477     fprintf(out, "\n");
02478
02479     /* Write data... */
02480     for (ip = 0; ip < atm->np; ip += ctl->atm_stride) {
02481
02482         /* Check time... */
02483         if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
02484             continue;
02485
02486         /* Write output... */
02487         fprintf(out, "%.2f %g %g", atm->time[ip], Z(atm->p[ip]),
02488                 atm->lon[ip], atm->lat[ip]);
02489         for (iq = 0; iq < ctl->nq; iq++) {
02490             fprintf(out, " ");
02491             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02492         }
02493         fprintf(out, "\n");
02494     }
02495 }

```

```

02496     /* Close file... */
02497     fclose(out);
02498 }
02499
02500 /* Write binary data... */
02501 else if (ctl->atm_type == 1) {
02502
02503     /* Create file... */
02504     if (!(out = fopen(filename, "w")))
02505         ERRMSG("Cannot create file!");
02506
02507     /* Write data... */
02508     FWRITE(&atm->np, int,
02509           1,
02510           out);
02511     FWRITE(atm->time, double,
02512           (size_t) atm->np,
02513           out);
02514     FWRITE(atm->p, double,
02515           (size_t) atm->np,
02516           out);
02517     FWRITE(atm->lon, double,
02518           (size_t) atm->np,
02519           out);
02520     FWRITE(atm->lat, double,
02521           (size_t) atm->np,
02522           out);
02523     for (iq = 0; iq < ctl->nq; iq++)
02524         FWRITE(atm->q[iq], double,
02525               (size_t) atm->np,
02526               out);
02527
02528     /* Close file... */
02529     fclose(out);
02530 }
02531
02532 /* Error... */
02533 else
02534     ERRMSG("Atmospheric data type not supported!");
02535 }

```

Here is the call graph for this function:



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

Write CSI data.

Definition at line 2539 of file libtrac.c.

```

02543     {
02544
02545     static FILE *in, *out;
02546
02547     static char line[LEN];
02548
02549     static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
02550           rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
02551
02552     static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
02553
02554     /* Init... */
02555     if (t == ctl->t_start) {

```

```

02556
02557 /* Check quantity index for mass... */
02558 if (ctl->qnt_m < 0)
02559     ERRMSG("Need quantity mass!");
02560
02561 /* Open observation data file... */
02562 printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
02563 if (!(in = fopen(ctl->csi_obsfile, "r")))
02564     ERRMSG("Cannot open file!");
02565
02566 /* Create new file... */
02567 printf("Write CSI data: %s\n", filename);
02568 if (!(out = fopen(filename, "w")))
02569     ERRMSG("Cannot create file!");
02570
02571 /* Write header... */
02572 fprintf(out,
02573         "# $1 = time [s]\n"
02574         "# $2 = number of hits (cx)\n"
02575         "# $3 = number of misses (cy)\n"
02576         "# $4 = number of false alarms (cz)\n"
02577         "# $5 = number of observations (cx + cy)\n"
02578         "# $6 = number of forecasts (cx + cz)\n"
02579         "# $7 = bias (forecasts/observations) [%%]\n"
02580         "# $8 = probability of detection (POD) [%%]\n"
02581         "# $9 = false alarm rate (FAR) [%%]\n"
02582         "# $10 = critical success index (CSI) [%%]\n\n");
02583 }
02584
02585 /* Set time interval... */
02586 t0 = t - 0.5 * ctl->dt_mod;
02587 t1 = t + 0.5 * ctl->dt_mod;
02588
02589 /* Initialize grid cells... */
02590 #pragma omp parallel for default(shared) private(ix,iy,iz)
02591 for (ix = 0; ix < ctl->csi_nx; ix++)
02592     for (iy = 0; iy < ctl->csi_ny; iy++)
02593         for (iz = 0; iz < ctl->csi_nz; iz++)
02594             modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
02595
02596 /* Read observation data... */
02597 while (fgets(line, LEN, in)) {
02598
02599     /* Read data... */
02600     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02601         5)
02602         continue;
02603
02604     /* Check time... */
02605     if (rt < t0)
02606         continue;
02607     if (rt > t1)
02608         break;
02609
02610     /* Calculate indices... */
02611     ix = (int) ((rlon - ctl->csi_lon0)
02612                / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02613     iy = (int) ((rlat - ctl->csi_lat0)
02614                / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02615     iz = (int) ((rz - ctl->csi_z0)
02616                / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02617
02618     /* Check indices... */
02619     if (ix < 0 || ix >= ctl->csi_nx ||
02620         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02621         continue;
02622
02623     /* Get mean observation index... */
02624     obsmean[ix][iy][iz] += robs;
02625     obscount[ix][iy][iz]++;
02626 }
02627
02628 /* Analyze model data... */
02629 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
02630 for (ip = 0; ip < atm->np; ip++) {
02631
02632     /* Check time... */
02633     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02634         continue;
02635
02636     /* Get indices... */
02637     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
02638                / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02639     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
02640                / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02641     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
02642                / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);

```



```

02643
02644     /* Check indices... */
02645     if (ix < 0 || ix >= ctl->csi_nx ||
02646         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02647         continue;
02648
02649     /* Get total mass in grid cell... */
02650     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02651 }
02652
02653 /* Analyze all grid cells... */
02654 #pragma omp parallel for default(shared) private(ix,iy,iz,dlon,dlat,lat,area)
02655 for (ix = 0; ix < ctl->csi_nx; ix++)
02656     for (iy = 0; iy < ctl->csi_ny; iy++)
02657         for (iz = 0; iz < ctl->csi_nz; iz++) {
02658
02659             /* Calculate mean observation index... */
02660             if (obscount[ix][iy][iz] > 0)
02661                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
02662
02663             /* Calculate column density... */
02664             if (modmean[ix][iy][iz] > 0) {
02665                 dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
02666                 dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
02667                 lat = ctl->csi_lat0 + dlat * (iy + 0.5);
02668                 area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
02669                     * cos(lat * M_PI / 180.);
02670                 modmean[ix][iy][iz] /= (1e6 * area);
02671             }
02672
02673             /* Calculate CSI... */
02674             if (obscount[ix][iy][iz] > 0) {
02675                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02676                     modmean[ix][iy][iz] >= ctl->csi_modmin)
02677                     cx++;
02678                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02679                     modmean[ix][iy][iz] < ctl->csi_modmin)
02680                     cy++;
02681                 else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
02682                     modmean[ix][iy][iz] >= ctl->csi_modmin)
02683                     cz++;
02684             }
02685         }
02686
02687     /* Write output... */
02688     if (fmod(t, ctl->csi_dt_out) == 0) {
02689
02690         /* Write... */
02691         fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
02692             t, cx, cy, cz, cx + cy, cx + cz,
02693             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
02694             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
02695             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
02696             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
02697
02698         /* Set counters to zero... */
02699         cx = cy = cz = 0;
02700     }
02701
02702     /* Close file... */
02703     if (t == ctl->t_stop)
02704         fclose(out);
02705 }

```

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

Write ensemble data.

Definition at line 2709 of file libtrac.c.

```

02713     {
02714
02715         static FILE *out;
02716
02717         static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
02718             t0, t1, x[NENS][3], xm[3];
02719
02720         static int ip, iq;
02721
02722         static size_t i, n;

```

```

02723
02724 /* Init... */
02725 if (t == ctl->t_start) {
02726
02727     /* Check quantities... */
02728     if (ctl->qnt_ens < 0)
02729         ERRMSG("Missing ensemble IDs!");
02730
02731     /* Create new file... */
02732     printf("Write ensemble data: %s\n", filename);
02733     if (!(out = fopen(filename, "w")))
02734         ERRMSG("Cannot create file!");
02735
02736     /* Write header... */
02737     fprintf(out,
02738             "# $1 = time [s]\n"
02739             "# $2 = altitude [km]\n"
02740             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02741     for (iq = 0; iq < ctl->nq; iq++)
02742         fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
02743                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02744     for (iq = 0; iq < ctl->nq; iq++)
02745         fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
02746                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02747     fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
02748 }
02749
02750 /* Set time interval... */
02751 t0 = t - 0.5 * ctl->dt_mod;
02752 t1 = t + 0.5 * ctl->dt_mod;
02753
02754 /* Init... */
02755 ens = GSL_NAN;
02756 n = 0;
02757
02758 /* Loop over air parcels... */
02759 for (ip = 0; ip < atm->np; ip++) {
02760
02761     /* Check time... */
02762     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02763         continue;
02764
02765     /* Check ensemble id... */
02766     if (atm->q[ctl->qnt_ens][ip] != ens) {
02767
02768         /* Write results... */
02769         if (n > 0) {
02770
02771             /* Get mean position... */
02772             xm[0] = xm[1] = xm[2] = 0;
02773             for (i = 0; i < n; i++) {
02774                 xm[0] += x[i][0] / (double) n;
02775                 xm[1] += x[i][1] / (double) n;
02776                 xm[2] += x[i][2] / (double) n;
02777             }
02778             cart2geo(xm, &dummy, &lon, &lat);
02779             fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
02780                     lat);
02781
02782             /* Get quantity statistics... */
02783             for (iq = 0; iq < ctl->nq; iq++) {
02784                 fprintf(out, " ");
02785                 fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02786             }
02787             for (iq = 0; iq < ctl->nq; iq++) {
02788                 fprintf(out, " ");
02789                 fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02790             }
02791             fprintf(out, " %lu\n", n);
02792         }
02793
02794         /* Init new ensemble... */
02795         ens = atm->q[ctl->qnt_ens][ip];
02796         n = 0;
02797     }
02798
02799     /* Save data... */
02800     p[n] = atm->p[ip];
02801     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
02802     for (iq = 0; iq < ctl->nq; iq++)
02803         q[iq][n] = atm->q[iq][ip];
02804     if ((++n) >= NENS)
02805         ERRMSG("Too many data points!");
02806 }
02807
02808 /* Write results... */
02809 if (n > 0) {

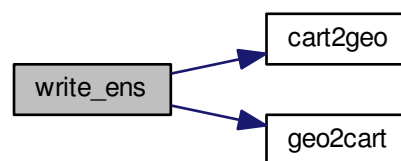
```

```

02810
02811     /* Get mean position... */
02812     xm[0] = xm[1] = xm[2] = 0;
02813     for (i = 0; i < n; i++) {
02814         xm[0] += x[i][0] / (double) n;
02815         xm[1] += x[i][1] / (double) n;
02816         xm[2] += x[i][2] / (double) n;
02817     }
02818     cart2geo(xm, &dummy, &lon, &lat);
02819     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
02820
02821     /* Get quantity statistics... */
02822     for (iq = 0; iq < ctl->nq; iq++) {
02823         fprintf(out, " ");
02824         fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02825     }
02826     for (iq = 0; iq < ctl->nq; iq++) {
02827         fprintf(out, " ");
02828         fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02829     }
02830     fprintf(out, " %lu\n", n);
02831 }
02832
02833 /* Close file... */
02834 if (t == ctl->t_stop)
02835     fclose(out);
02836 }

```

Here is the call graph for this function:



5.19.2.36 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 2840 of file `libtrac.c`.

```

02846     {
02847
02848     FILE *in, *out;
02849
02850     char line[LEN];
02851
02852     static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
02853         area, rho_air, press, temp, cd, vmr, t0, t1, r;
02854
02855     static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec;
02856
02857     /* Check dimensions... */
02858     if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
02859         ERRMSG("Grid dimensions too large!");
02860
02861     /* Set time interval for output... */
02862     t0 = t - 0.5 * ctl->dt_mod;
02863     t1 = t + 0.5 * ctl->dt_mod;
02864
02865     /* Set grid box size... */
02866     dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
02867     dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;

```

```

02868     dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
02869
02870     /* Initialize grid... */
02871 #pragma omp parallel for default(shared) private(ix,iy,iz)
02872     for (ix = 0; ix < ctl->grid_nx; ix++)
02873         for (iy = 0; iy < ctl->grid_ny; iy++)
02874             for (iz = 0; iz < ctl->grid_nz; iz++) {
02875                 mass[ix][iy][iz] = 0;
02876                 np[ix][iy][iz] = 0;
02877             }
02878
02879     /* Average data... */
02880 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
02881     for (ip = 0; ip < atm->np; ip++)
02882         if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
02883
02884             /* Get index... */
02885             ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
02886             iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
02887             iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
02888
02889             /* Check indices... */
02890             if (ix < 0 || ix >= ctl->grid_nx ||
02891                 iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
02892                 continue;
02893
02894             /* Add mass... */
02895             if (ctl->qnt_m >= 0)
02896                 mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02897             np[ix][iy][iz]++;
02898         }
02899
02900     /* Check if gnuplot output is requested... */
02901     if (ctl->grid_gpfile[0] != '-') {
02902
02903         /* Write info... */
02904         printf("Plot grid data: %s.png\n", filename);
02905
02906         /* Create gnuplot pipe... */
02907         if (!(out = popen("gnuplot", "w")))
02908             ERRMSG("Cannot create pipe to gnuplot!");
02909
02910         /* Set plot filename... */
02911         fprintf(out, "set out \"%s.png\"\n", filename);
02912
02913         /* Set time string... */
02914         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02915         fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02916             year, mon, day, hour, min);
02917
02918         /* Dump gnuplot file to pipe... */
02919         if (!(in = fopen(ctl->grid_gpfile, "r")))
02920             ERRMSG("Cannot open file!");
02921         while (fgets(line, LEN, in))
02922             fprintf(out, "%s", line);
02923         fclose(in);
02924     }
02925
02926     else {
02927
02928         /* Write info... */
02929         printf("Write grid data: %s\n", filename);
02930
02931         /* Create file... */
02932         if (!(out = fopen(filename, "w")))
02933             ERRMSG("Cannot create file!");
02934     }
02935
02936     /* Write header... */
02937     fprintf(out,
02938         "# $1 = time [s]\n"
02939         "# $2 = altitude [km]\n"
02940         "# $3 = longitude [deg]\n"
02941         "# $4 = latitude [deg]\n"
02942         "# $5 = surface area [km^2]\n"
02943         "# $6 = layer width [km]\n"
02944         "# $7 = number of particles [l]\n"
02945         "# $8 = column density [kg/m^2]\n"
02946         "# $9 = volume mixing ratio [l]\n\n");
02947
02948     /* Write data... */
02949     for (ix = 0; ix < ctl->grid_nx; ix++) {
02950         if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
02951             fprintf(out, "\n");
02952         for (iy = 0; iy < ctl->grid_ny; iy++) {
02953             if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
02954                 fprintf(out, "\n");

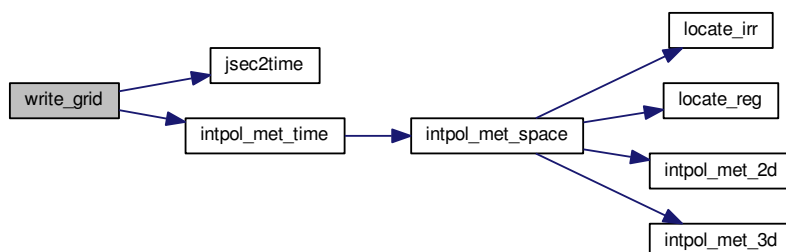
```

```

02955     for (iz = 0; iz < ctl->grid_nz; iz++)
02956     if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
02957
02958         /* Set coordinates... */
02959         z = ctl->grid_z0 + dz * (iz + 0.5);
02960         lon = ctl->grid_lon0 + dlon * (ix + 0.5);
02961         lat = ctl->grid_lat0 + dlat * (iy + 0.5);
02962
02963         /* Get pressure and temperature... */
02964         press = P(z);
02965         intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
02966                        NULL, &temp, NULL, NULL, NULL, NULL, NULL, NULL);
02967
02968         /* Calculate surface area... */
02969         area = dlat * dlon * SQR(RE * M_PI / 180.)
02970             * cos(lat * M_PI / 180.);
02971
02972         /* Calculate column density... */
02973         cd = mass[ix][iy][iz] / (1e6 * area);
02974
02975         /* Calculate volume mixing ratio... */
02976         rho_air = 100. * press / (RA * temp);
02977         vmr = MA / ctl->molmass * mass[ix][iy][iz]
02978             / (rho_air * 1e6 * area * 1e3 * dz);
02979
02980         /* Write output... */
02981         fprintf(out, "%.2f %g %g %g %g %g %d %g %g\n",
02982              t, z, lon, lat, area, dz, np[ix][iy][iz], cd, vmr);
02983     }
02984 }
02985 }
02986
02987 /* Close file... */
02988 fclose(out);
02989 }

```

Here is the call graph for this function:



5.19.2.37 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 2993 of file libtrac.c.

```

02999     {
03000
03001     static FILE *in, *out;
03002
03003     static char line[LEN];
03004
03005     static double mass[GX][GY][GZ], obsmean[GX][GY], obsmean2[GX][GY], rt, rz,
03006         rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp,
03007         rho_air, vmr, h2o, o3;
03008
03009     static int obscount[GX][GY], ip, ix, iy, iz, okay;
03010

```

```

03011  /* Init... */
03012  if (t == ctl->t_start) {
03013
03014      /* Check quantity index for mass... */
03015      if (ctl->qnt_m < 0)
03016          ERRMSG("Need quantity mass!");
03017
03018      /* Check dimensions... */
03019      if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
03020          ERRMSG("Grid dimensions too large!");
03021
03022      /* Open observation data file... */
03023      printf("Read profile observation data: %s\n", ctl->prof_obsfile);
03024      if (!(in = fopen(ctl->prof_obsfile, "r")))
03025          ERRMSG("Cannot open file!");
03026
03027      /* Create new output file... */
03028      printf("Write profile data: %s\n", filename);
03029      if (!(out = fopen(filename, "w")))
03030          ERRMSG("Cannot create file!");
03031
03032      /* Write header... */
03033      fprintf(out,
03034              "# $1 = time [s]\n"
03035              "# $2 = altitude [km]\n"
03036              "# $3 = longitude [deg]\n"
03037              "# $4 = latitude [deg]\n"
03038              "# $5 = pressure [hPa]\n"
03039              "# $6 = temperature [K]\n"
03040              "# $7 = volume mixing ratio [l]\n"
03041              "# $8 = H2O volume mixing ratio [l]\n"
03042              "# $9 = O3 volume mixing ratio [l]\n"
03043              "# $10 = observed BT index (mean) [K]\n"
03044              "# $11 = observed BT index (sigma) [K]\n");
03045
03046      /* Set grid box size... */
03047      dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
03048      dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
03049      dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
03050  }
03051
03052      /* Set time interval... */
03053      t0 = t - 0.5 * ctl->dt_mod;
03054      t1 = t + 0.5 * ctl->dt_mod;
03055
03056      /* Initialize... */
03057      #pragma omp parallel for default(shared) private(ix,iy,iz)
03058      for (ix = 0; ix < ctl->prof_nx; ix++)
03059          for (iy = 0; iy < ctl->prof_ny; iy++) {
03060              obsmean[ix][iy] = 0;
03061              obsmean2[ix][iy] = 0;
03062              obscount[ix][iy] = 0;
03063              for (iz = 0; iz < ctl->prof_nz; iz++)
03064                  mass[ix][iy][iz] = 0;
03065          }
03066
03067      /* Read observation data... */
03068      while (fgets(line, LEN, in)) {
03069
03070          /* Read data... */
03071          if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rlat, &robs) !=
03072              5)
03073              continue;
03074
03075          /* Check time... */
03076          if (rt < t0)
03077              continue;
03078          if (rt > t1)
03079              break;
03080
03081          /* Calculate indices... */
03082          ix = (int) ((rln - ctl->prof_lon0) / dlon);
03083          iy = (int) ((rlat - ctl->prof_lat0) / dlat);
03084
03085          /* Check indices... */
03086          if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
03087              continue;
03088
03089          /* Get mean observation index... */
03090          obsmean[ix][iy] += robs;
03091          obsmean2[ix][iy] += SQR(robs);
03092          obscount[ix][iy]++;
03093      }
03094
03095      /* Analyze model data... */
03096      #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
03097      for (ip = 0; ip < atm->np; ip++) {

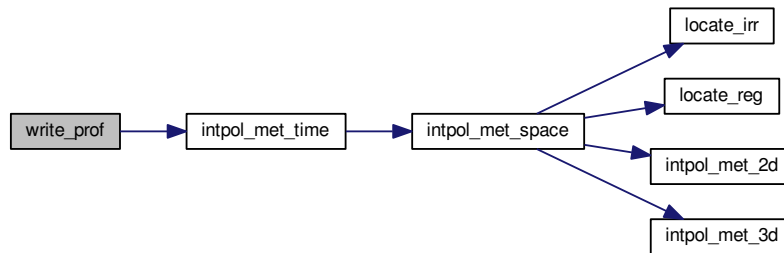
```

```

03098
03099     /* Check time... */
03100     if (atm->time[ip] < t0 || atm->time[ip] > t1)
03101         continue;
03102
03103     /* Get indices... */
03104     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
03105     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
03106     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
03107
03108     /* Check indices... */
03109     if (ix < 0 || ix >= ctl->prof_nx ||
03110         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
03111         continue;
03112
03113     /* Get total mass in grid cell... */
03114     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
03115 }
03116
03117 /* Extract profiles... */
03118 for (ix = 0; ix < ctl->prof_nx; ix++)
03119     for (iy = 0; iy < ctl->prof_ny; iy++)
03120         if (obscount[ix][iy] > 0) {
03121
03122             /* Check profile... */
03123             okay = 0;
03124             for (iz = 0; iz < ctl->prof_nz; iz++)
03125                 if (mass[ix][iy][iz] > 0) {
03126                     okay = 1;
03127                     break;
03128                 }
03129             if (!okay)
03130                 continue;
03131
03132             /* Write output... */
03133             fprintf(out, "\n");
03134
03135             /* Loop over altitudes... */
03136             for (iz = 0; iz < ctl->prof_nz; iz++) {
03137
03138                 /* Set coordinates... */
03139                 z = ctl->prof_z0 + dz * (iz + 0.5);
03140                 lon = ctl->prof_lon0 + dlon * (ix + 0.5);
03141                 lat = ctl->prof_lat0 + dlat * (iy + 0.5);
03142
03143                 /* Get pressure and temperature... */
03144                 press = P(z);
03145                 intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
03146                     NULL, &temp, NULL, NULL, NULL, NULL, &h2o, &o3);
03147
03148                 /* Calculate surface area... */
03149                 area = dlat * dlon * SQR(M_PI * RE / 180.)
03150                     * cos(lat * M_PI / 180.);
03151
03152                 /* Calculate volume mixing ratio... */
03153                 rho_air = 100. * press / (RA * temp);
03154                 vmr = MA / ctl->molmass * mass[ix][iy][iz]
03155                     / (rho_air * area * dz * 1e9);
03156
03157                 /* Write output... */
03158                 fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
03159                     t, z, lon, lat, press, temp, vmr, h2o, o3,
03160                     obsmean[ix][iy] / obscount[ix][iy],
03161                     sqrt(obsmean2[ix][iy] / obscount[ix][iy]
03162                         - SQR(obsmean[ix][iy] / obscount[ix][iy])));
03163             }
03164         }
03165
03166     /* Close file... */
03167     if (t == ctl->t_stop)
03168         fclose(out);
03169 }

```

Here is the call graph for this function:



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

Write station data.

Definition at line 3173 of file libtrac.c.

```

03177     {
03178
03179     static FILE *out;
03180
03181     static double rmax2, t0, t1, x0[3], x1[3];
03182
03183     /* Init... */
03184     if (t == ctl->t_start) {
03185
03186         /* Write info... */
03187         printf("Write station data: %s\n", filename);
03188
03189         /* Create new file... */
03190         if (!(out = fopen(filename, "w")))
03191             ERRMSG("Cannot create file!");
03192
03193         /* Write header... */
03194         fprintf(out,
03195             "# $1 = time [s]\n"
03196             "# $2 = altitude [km]\n"
03197             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03198         for (int iq = 0; iq < ctl->nq; iq++)
03199             fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
03200                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
03201         fprintf(out, "\n");
03202
03203         /* Set geolocation and search radius... */
03204         geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
03205         rmax2 = SQR(ctl->stat_r);
03206     }
03207
03208     /* Set time interval for output... */
03209     t0 = t - 0.5 * ctl->dt_mod;
03210     t1 = t + 0.5 * ctl->dt_mod;
03211
03212     /* Loop over air parcels... */
03213     for (int ip = 0; ip < atm->np; ip++) {
03214
03215         /* Check time... */
03216         if (atm->time[ip] < t0 || atm->time[ip] > t1)
03217             continue;
03218
03219         /* Check station flag... */
03220         if (ctl->qnt_stat >= 0)
03221             if (atm->q[ctl->qnt_stat][ip])
03222                 continue;
03223
03224         /* Get Cartesian coordinates... */
03225         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
03226     }
  
```



```

03227     /* Check horizontal distance... */
03228     if (DIST2(x0, x1) > rmax2)
03229         continue;
03230
03231     /* Set station flag... */
03232     if (ctl->qnt_stat >= 0)
03233         atm->q[ctl->qnt_stat][ip] = 1;
03234
03235     /* Write data... */
03236     fprintf(out, "%.2f %g %g %g",
03237             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
03238     for (int iq = 0; iq < ctl->nq; iq++) {
03239         fprintf(out, " ");
03240         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
03241     }
03242     fprintf(out, "\n");
03243 }
03244
03245 /* Close file... */
03246 if (t == ctl->t_stop)
03247     fclose(out);
03248 }

```

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) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /*****
00028
00029 void cart2geo(
00030     double *x,
00031     double *z,
00032     double *lon,
00033     double *lat) {
00034
00035     double radius = NORM(x);
00036     *lat = asin(x[2] / radius) * 180 / M_PI;
00037     *lon = atan2(x[1], x[0]) * 180 / M_PI;
00038     *z = radius - RE;
00039 }
00040
00041 /*****
00042
00043 static double clim_hno3_secs[12] = {

```

```

00044 1209600.00, 3888000.00, 6393600.00,
00045 9072000.00, 11664000.00, 14342400.00,
00046 16934400.00, 19612800.00, 22291200.00,
00047 24883200.00, 27561600.00, 30153600.00
00048 };
00049
00050 #ifdef _OPENACC
00051 #pragma acc declare copyin(clim_hno3_secs)
00052 #endif
00053
00054 static double clim_hno3_lats[18] = {
00055 -85, -75, -65, -55, -45, -35, -25, -15, -5,
00056 5, 15, 25, 35, 45, 55, 65, 75, 85
00057 };
00058
00059 #ifdef _OPENACC
00060 #pragma acc declare copyin(clim_hno3_lats)
00061 #endif
00062
00063 static double clim_hno3_ps[10] = {
00064 4.64159, 6.81292, 10, 14.678, 21.5443,
00065 31.6228, 46.4159, 68.1292, 100, 146.78
00066 };
00067
00068 #ifdef _OPENACC
00069 #pragma acc declare copyin(clim_hno3_ps)
00070 #endif
00071
00072 static double clim_hno3_var[12][18][10] = {
00073 {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00074 {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00075 {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
00076 {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00077 {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
00078 {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
00079 {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00080 {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
00081 {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
00082 {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00083 {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
00084 {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00085 {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
00086 {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00087 {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00088 {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00089 {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
00090 {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
00091 {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00092 {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
00093 {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
00094 {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00095 {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
00096 {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
00097 {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
00098 {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
00099 {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
00100 {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
00101 {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00102 {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
00103 {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
00104 {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
00105 {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
00106 {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
00107 {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
00108 {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
00109 {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00110 {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
00111 {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
00112 {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
00113 {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
00114 {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
00115 {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
00116 {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
00117 {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
00118 {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
00119 {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
00120 {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
00121 {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
00122 {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
00123 {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
00124 {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
00125 {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
00126 {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
00127 {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
00128 {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00129 {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
00130 {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},

```

```

00131 {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
00132 {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
00133 {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
00134 {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
00135 {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
00136 {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
00137 {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
00138 {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
00139 {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
00140 {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
00141 {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
00142 {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
00143 {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
00144 {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62},
00145 {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
00146 {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
00147 {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
00148 {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
00149 {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
00150 {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
00151 {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
00152 {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
00153 {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
00154 {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
00155 {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
00156 {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
00157 {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
00158 {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
00159 {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
00160 {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
00161 {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
00162 {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6},
00163 {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
00164 {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
00165 {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
00166 {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
00167 {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
00168 {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
00169 {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
00170 {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
00171 {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
00172 {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
00173 {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
00174 {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
00175 {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
00176 {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
00177 {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
00178 {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
00179 {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
00180 {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91},
00181 {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
00182 {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
00183 {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
00184 {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
00185 {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
00186 {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
00187 {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
00188 {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
00189 {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
00190 {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
00191 {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
00192 {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
00193 {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
00194 {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
00195 {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
00196 {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
00197 {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
00198 {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62},
00199 {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
00200 {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
00201 {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
00202 {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
00203 {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
00204 {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
00205 {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
00206 {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
00207 {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
00208 {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
00209 {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
00210 {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
00211 {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
00212 {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
00213 {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
00214 {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
00215 {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
00216 {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55},
00217 {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},

```

```

00218 {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
00219 {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
00220 {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
00221 {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
00222 {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
00223 {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
00224 {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
00225 {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
00226 {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
00227 {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
00228 {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
00229 {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
00230 {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
00231 {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
00232 {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
00233 {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
00234 {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65},
00235 {0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
00236 {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
00237 {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
00238 {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
00239 {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
00240 {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
00241 {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
00242 {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
00243 {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
00244 {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
00245 {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
00246 {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
00247 {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
00248 {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
00249 {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
00250 {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
00251 {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
00252 {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8},
00253 {0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
00254 {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
00255 {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
00256 {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
00257 {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
00258 {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
00259 {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
00260 {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
00261 {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
00262 {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
00263 {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
00264 {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
00265 {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
00266 {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
00267 {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
00268 {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
00269 {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
00270 {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05},
00271 {0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
00272 {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
00273 {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
00274 {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
00275 {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
00276 {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
00277 {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
00278 {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
00279 {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
00280 {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
00281 {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
00282 {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
00283 {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
00284 {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
00285 {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
00286 {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
00287 {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
00288 {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}
00289 };
00290
00291 #ifndef _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
00303     /* Get indices... */
00304     int isec = locate_irr(clim_hno3_secs, 12, sec);

```

```

00305     int ilat = locate_reg(clim_hno3_lats, 18, lat);
00306     int ip = locate_irr(clim_hno3_ps, 10, p);
00307
00308     /* Interpolate... */
00309     double aux00 = LIN(clim_hno3_ps[ip], clim_hno3_var[isec][ilat][ip],
00310                       clim_hno3_ps[ip + 1], clim_hno3_var[isec][ilat][ip + 1],
00311                       p);
00312     double aux01 = LIN(clim_hno3_ps[ip], clim_hno3_var[isec][ilat + 1][ip],
00313                       clim_hno3_ps[ip + 1],
00314                       clim_hno3_var[isec][ilat + 1][ip + 1], p);
00315     double aux10 = LIN(clim_hno3_ps[ip], clim_hno3_var[isec + 1][ilat][ip],
00316                       clim_hno3_ps[ip + 1],
00317                       clim_hno3_var[isec + 1][ilat][ip + 1], p);
00318     double aux11 = LIN(clim_hno3_ps[ip], clim_hno3_var[isec + 1][ilat + 1][ip],
00319                       clim_hno3_ps[ip + 1],
00320                       clim_hno3_var[isec + 1][ilat + 1][ip + 1],
00321                       p);
00322     aux00 =
00323         LIN(clim_hno3_lats[ilat], aux00, clim_hno3_lats[ilat + 1], aux01, lat);
00324     aux11 =
00325         LIN(clim_hno3_lats[ilat], aux10, clim_hno3_lats[ilat + 1], aux11, lat);
00326     return LIN(clim_hno3_secs[isec], aux00, clim_hno3_secs[isec + 1], aux11,
00327               sec);
00328 }
00329
00330 /*****
00331
00332 static double clim_tropo_doys[12]
00333 = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00334
00335 #ifdef _OPENACC
00336 #pragma acc declare copyin(clim_tropo_doys)
00337 #endif
00338
00339 static double clim_tropo_lats[73]
00340 = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
00341     -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
00342     -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
00343     -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
00344     15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
00345     45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
00346     75, 77.5, 80, 82.5, 85, 87.5, 90
00347 };
00348
00349 #ifdef _OPENACC
00350 #pragma acc declare copyin(clim_tropo_lats)
00351 #endif
00352
00353 static double clim_tropo_tps[12][73]
00354 = { { 324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
00355     297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
00356     175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
00357     99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
00358     98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
00359     152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
00360     277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
00361     275.3, 275.6, 275.4, 274.1, 273.5},
00362 { 337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
00363     300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
00364     150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
00365     98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
00366     98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
00367     220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
00368     284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
00369     287.5, 286.2, 285.8},
00370 { 335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
00371     297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
00372     161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
00373     100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
00374     99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
00375     186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
00376     279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
00377     304.3, 304.9, 306, 306.6, 306.2, 306},
00378 { 306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
00379     290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
00380     195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
00381     102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
00382     99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
00383     148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
00384     263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
00385     315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
00386 { 266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
00387     260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
00388     205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
00389     101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
00390     102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
00391     165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,

```

```

00392 273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
00393 325.3, 325.8, 325.8},
00394 {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
00395 222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
00396 228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
00397 105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
00398 106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
00399 127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
00400 251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
00401 308.5, 312.2, 313.1, 313.3},
00402 {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
00403 187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
00404 235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
00405 110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
00406 111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
00407 117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
00408 224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
00409 275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
00410 {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
00411 185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
00412 233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
00413 110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
00414 112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
00415 120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
00416 230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
00417 278.2, 282.6, 287.4, 290.9, 292.5, 293},
00418 {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
00419 183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
00420 243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
00421 114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
00422 110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
00423 114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
00424 203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
00425 276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
00426 {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
00427 215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
00428 237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
00429 111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
00430 106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
00431 112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
00432 206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
00433 279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
00434 305.1},
00435 {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
00436 253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
00437 223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
00438 108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
00439 102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
00440 109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
00441 241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
00442 286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
00443 {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
00444 284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
00445 175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
00446 100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
00447 100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
00448 186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
00449 280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
00450 281.7, 281.1, 281.2}
00451 };
00452
00453 #ifdef _OPENACC
00454 #pragma acc declare copyin(clim_tropo_tps)
00455 #endif
00456
00457 double clim_tropo(
00458     double t,
00459     double lat) {
00460
00461     /* Get day of year... */
00462     double doy = FMOD(t / 86400., 365.25);
00463     while (doy < 0)
00464         doy += 365.25;
00465
00466     /* Get indices... */
00467     int ilat = locate_reg(clim_tropo_lats, 73, lat);
00468     int imon = locate_irr(clim_tropo_doys, 12, doy);
00469
00470     /* Interpolate... */
00471     double p0 = LIN(clim_tropo_lats[ilat], clim_tropo_tps[imon][ilat],
00472                    clim_tropo_lats[ilat + 1], clim_tropo_tps[imon][ilat + 1],
00473                    lat);
00474     double p1 = LIN(clim_tropo_lats[ilat], clim_tropo_tps[imon + 1][ilat],
00475                    clim_tropo_lats[ilat + 1],
00476                    clim_tropo_tps[imon + 1][ilat + 1],
00477                    lat);
00478     return LIN(clim_tropo_doys[imon], p0, clim_tropo_doys[imon + 1], p1, doy);

```

```

00479 }
00480
00481 /*****
00482
00483 void day2doy(
00484     int year,
00485     int mon,
00486     int day,
00487     int *doy) {
00488
00489     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00490     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00491
00492     /* Get day of year... */
00493     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
00494         *doy = d0l[mon - 1] + day - 1;
00495     else
00496         *doy = d0[mon - 1] + day - 1;
00497 }
00498
00499 /*****
00500
00501 void doy2day(
00502     int year,
00503     int doy,
00504     int *mon,
00505     int *day) {
00506
00507     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00508     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00509     int i;
00510
00511     /* Get month and day... */
00512     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
00513         for (i = 11; i >= 0; i--)
00514             if (d0l[i] <= doy)
00515                 break;
00516         *mon = i + 1;
00517         *day = doy - d0l[i] + 1;
00518     } else {
00519         for (i = 11; i >= 0; i--)
00520             if (d0[i] <= doy)
00521                 break;
00522         *mon = i + 1;
00523         *day = doy - d0[i] + 1;
00524     }
00525 }
00526
00527 /*****
00528
00529 void geo2cart(
00530     double z,
00531     double lon,
00532     double lat,
00533     double *x) {
00534
00535     double radius = z + RE;
00536     x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
00537     x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
00538     x[2] = radius * sin(lat / 180 * M_PI);
00539 }
00540
00541 /*****
00542
00543 void get_met(
00544     ctl_t *ctl,
00545     char *metbase,
00546     double t,
00547     met_t **met0,
00548     met_t **met1) {
00549
00550     static int init, ip, ix, iy;
00551
00552     met_t *mets;
00553
00554     char filename[LEN];
00555
00556     /* Init... */
00557     if (t == ctl->t_start || !init) {
00558         init = 1;
00559
00560         get_met_help(t, -1, metbase, ctl->dt_met, filename);
00561         if (!read_met(ctl, filename, *met0))
00562             ERRMSG("Cannot open file!");
00563
00564         get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
dt_met, filename);

```

```

00565     if (!read_met(ctl, filename, *met1))
00566         ERRMSG("Cannot open file!");
00567 #ifdef _OPENACC
00568     met_t *met0up = *met0;
00569     met_t *met1up = *met1;
00570 #pragma acc update device(met0up[:1],met1up[:1])
00571 #endif
00572 }
00573
00574 /* Read new data for forward trajectories... */
00575 if (t > (*met1)->time && ctl->direction == 1) {
00576     mets = *met1;
00577     *met1 = *met0;
00578     *met0 = mets;
00579     get_met_help(t, 1, metbase, ctl->dt_met, filename);
00580     if (!read_met(ctl, filename, *met1))
00581         ERRMSG("Cannot open file!");
00582 #ifdef _OPENACC
00583     met_t *met1up = *met1;
00584 #pragma acc update device(met1up[:1])
00585 #endif
00586 }
00587
00588 /* Read new data for backward trajectories... */
00589 if (t < (*met0)->time && ctl->direction == -1) {
00590     mets = *met1;
00591     *met1 = *met0;
00592     *met0 = mets;
00593     get_met_help(t, -1, metbase, ctl->dt_met, filename);
00594     if (!read_met(ctl, filename, *met0))
00595         ERRMSG("Cannot open file!");
00596 #ifdef _OPENACC
00597     met_t *met0up = *met0;
00598 #pragma acc update device(met0up[:1])
00599 #endif
00600 }
00601
00602 /* Check that grids are consistent... */
00603 if ((*met0)->nx != (*met1)->nx
00604     || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
00605     ERRMSG("Meteo grid dimensions do not match!");
00606 for (ix = 0; ix < (*met0)->nx; ix++)
00607     if ((*met0)->lon[ix] != (*met1)->lon[ix])
00608         ERRMSG("Meteo grid longitudes do not match!");
00609 for (iy = 0; iy < (*met0)->ny; iy++)
00610     if ((*met0)->lat[iy] != (*met1)->lat[iy])
00611         ERRMSG("Meteo grid latitudes do not match!");
00612 for (ip = 0; ip < (*met0)->np; ip++)
00613     if ((*met0)->p[ip] != (*met1)->p[ip])
00614         ERRMSG("Meteo grid pressure levels do not match!");
00615 }
00616
00617 /*****
00618
00619 void get_met_help(
00620     double t,
00621     int direct,
00622     char *metbase,
00623     double dt_met,
00624     char *filename) {
00625
00626     char repl[LEN];
00627
00628     double t6, r;
00629
00630     int year, mon, day, hour, min, sec;
00631
00632     /* Round time to fixed intervals... */
00633     if (direct == -1)
00634         t6 = floor(t / dt_met) * dt_met;
00635     else
00636         t6 = ceil(t / dt_met) * dt_met;
00637
00638     /* Decode time... */
00639     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00640
00641     /* Set filename... */
00642     sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
00643     sprintf(repl, "%d", year);
00644     get_met_replace(filename, "YYYY", repl);
00645     sprintf(repl, "%02d", mon);
00646     get_met_replace(filename, "MM", repl);
00647     sprintf(repl, "%02d", day);
00648     get_met_replace(filename, "DD", repl);
00649     sprintf(repl, "%02d", hour);
00650     get_met_replace(filename, "HH", repl);
00651 }

```



```

00652
00653 /*****
00654
00655 void get_met_replace(
00656     char *orig,
00657     char *search,
00658     char *repl) {
00659
00660     char buffer[LEN], *ch;
00661
00662     /* Iterate... */
00663     for (int i = 0; i < 3; i++) {
00664
00665         /* Replace substring... */
00666         if (!(ch = strstr(orig, search)))
00667             return;
00668         strncpy(buffer, orig, (size_t) (ch - orig));
00669         buffer[ch - orig] = 0;
00670         sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
00671         orig[0] = 0;
00672         strcpy(orig, buffer);
00673     }
00674 }
00675
00676 /*****
00677
00678 double intpol_met_2d(
00679     double array[EX][EY],
00680     int ix,
00681     int iy,
00682     double wx,
00683     double wy) {
00684
00685     /* Set variables... */
00686     double aux00 = array[ix][iy];
00687     double aux01 = array[ix][iy + 1];
00688     double aux10 = array[ix + 1][iy];
00689     double aux11 = array[ix + 1][iy + 1];
00690
00691     /* Interpolate horizontally... */
00692     aux00 = wy * (aux00 - aux01) + aux01;
00693     aux11 = wy * (aux10 - aux11) + aux11;
00694     return wx * (aux00 - aux11) + aux11;
00695 }
00696
00697 /*****
00698
00699 double intpol_met_3d(
00700     float array[EX][EY][EP],
00701     int ip,
00702     int ix,
00703     int iy,
00704     double wp,
00705     double wx,
00706     double wy) {
00707
00708     /* Interpolate vertically... */
00709     double aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00710         + array[ix][iy][ip + 1];
00711     double aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
00712         + array[ix][iy + 1][ip + 1];
00713     double aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
00714         + array[ix + 1][iy][ip + 1];
00715     double aux11 =
00716         wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
00717         + array[ix + 1][iy + 1][ip + 1];
00718
00719     /* Interpolate horizontally... */
00720     aux00 = wy * (aux00 - aux01) + aux01;
00721     aux11 = wy * (aux10 - aux11) + aux11;
00722     return wx * (aux00 - aux11) + aux11;
00723 }
00724
00725 /*****
00726
00727 void intpol_met_space(
00728     met_t * met,
00729     double p,
00730     double lon,
00731     double lat,
00732     double *ps,
00733     double *pt,
00734     double *z,
00735     double *t,
00736     double *u,
00737     double *v,
00738     double *w,

```

```

00739 double *pv,
00740 double *h2o,
00741 double *o3) {
00742
00743 /* Check longitude... */
00744 if (met->lon[met->nx - 1] > 180 && lon < 0)
00745     lon += 360;
00746
00747 /* Get indices... */
00748 int ip = locate_irr(met->p, met->np, p);
00749 int ix = locate_reg(met->lon, met->nx, lon);
00750 int iy = locate_reg(met->lat, met->ny, lat);
00751
00752 /* Get weights... */
00753 double wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
00754 double wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
00755 double wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);
00756
00757 /* Interpolate... */
00758 if (ps != NULL)
00759     *ps = intpol_met_2d(met->ps, ix, iy, wx, wy);
00760 if (pt != NULL)
00761     *pt = intpol_met_2d(met->pt, ix, iy, wx, wy);
00762 if (z != NULL)
00763     *z = intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy);
00764 if (t != NULL)
00765     *t = intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy);
00766 if (u != NULL)
00767     *u = intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy);
00768 if (v != NULL)
00769     *v = intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy);
00770 if (w != NULL)
00771     *w = intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy);
00772 if (pv != NULL)
00773     *pv = intpol_met_3d(met->pv, ip, ix, iy, wp, wx, wy);
00774 if (h2o != NULL)
00775     *h2o = intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy);
00776 if (o3 != NULL)
00777     *o3 = intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy);
00778 }
00779
00780 /*****
00781
00782 void intpol_met_time(
00783     met_t * met0,
00784     met_t * met1,
00785     double ts,
00786     double p,
00787     double lon,
00788     double lat,
00789     double *ps,
00790     double *pt,
00791     double *z,
00792     double *t,
00793     double *u,
00794     double *v,
00795     double *w,
00796     double *pv,
00797     double *h2o,
00798     double *o3) {
00799
00800     double h2o0, h2o1, o30, o31, ps0, ps1, pt0, pt1, pv0, pv1, t0, t1, u0, u1,
00801         v0, v1, w0, w1, wt, z0, z1;
00802
00803 /* Spatial interpolation... */
00804 intpol_met_space(met0, p, lon, lat,
00805                 ps == NULL ? NULL : &ps0,
00806                 pt == NULL ? NULL : &pt0,
00807                 z == NULL ? NULL : &z0,
00808                 t == NULL ? NULL : &t0,
00809                 u == NULL ? NULL : &u0,
00810                 v == NULL ? NULL : &v0,
00811                 w == NULL ? NULL : &w0,
00812                 pv == NULL ? NULL : &pv0,
00813                 h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00814 intpol_met_space(met1, p, lon, lat,
00815                 ps == NULL ? NULL : &ps1,
00816                 pt == NULL ? NULL : &pt1,
00817                 z == NULL ? NULL : &z1,
00818                 t == NULL ? NULL : &t1,
00819                 u == NULL ? NULL : &u1,
00820                 v == NULL ? NULL : &v1,
00821                 w == NULL ? NULL : &w1,
00822                 pv == NULL ? NULL : &pv1,
00823                 h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00824
00825 /* Get weighting factor... */

```

```

00826     wt = (met1->time - ts) / (met1->time - met0->time);
00827
00828     /* Interpolate... */
00829     if (ps != NULL)
00830         *ps = wt * (ps0 - ps1) + ps1;
00831     if (pt != NULL)
00832         *pt = wt * (pt0 - pt1) + pt1;
00833     if (z != NULL)
00834         *z = wt * (z0 - z1) + z1;
00835     if (t != NULL)
00836         *t = wt * (t0 - t1) + t1;
00837     if (u != NULL)
00838         *u = wt * (u0 - u1) + u1;
00839     if (v != NULL)
00840         *v = wt * (v0 - v1) + v1;
00841     if (w != NULL)
00842         *w = wt * (w0 - w1) + w1;
00843     if (pv != NULL)
00844         *pv = wt * (pv0 - pv1) + pv1;
00845     if (h2o != NULL)
00846         *h2o = wt * (h2o0 - h2o1) + h2o1;
00847     if (o3 != NULL)
00848         *o3 = wt * (o30 - o31) + o31;
00849 }
00850
00851 /*****
00852
00853 void jsec2time(
00854     double jsec,
00855     int *year,
00856     int *mon,
00857     int *day,
00858     int *hour,
00859     int *min,
00860     int *sec,
00861     double *remain) {
00862
00863     struct tm t0, *t1;
00864
00865     t0.tm_year = 100;
00866     t0.tm_mon = 0;
00867     t0.tm_mday = 1;
00868     t0.tm_hour = 0;
00869     t0.tm_min = 0;
00870     t0.tm_sec = 0;
00871
00872     time_t jsec0 = (time_t) jsec + timegm(&t0);
00873     t1 = gmtime(&jsec0);
00874
00875     *year = t1->tm_year + 1900;
00876     *mon = t1->tm_mon + 1;
00877     *day = t1->tm_mday;
00878     *hour = t1->tm_hour;
00879     *min = t1->tm_min;
00880     *sec = t1->tm_sec;
00881     *remain = jsec - floor(jsec);
00882 }
00883
00884 /*****
00885
00886 int locate_irr(
00887     double *xx,
00888     int n,
00889     double x) {
00890
00891     int ilo = 0;
00892     int ihi = n - 1;
00893     int i = (ihi + ilo) >> 1;
00894
00895     if (xx[i] < xx[i + 1])
00896         while (ihi > ilo + 1) {
00897             i = (ihi + ilo) >> 1;
00898             if (xx[i] > x)
00899                 ihi = i;
00900             else
00901                 ilo = i;
00902         } else
00903         while (ihi > ilo + 1) {
00904             i = (ihi + ilo) >> 1;
00905             if (xx[i] <= x)
00906                 ihi = i;
00907             else
00908                 ilo = i;
00909         }
00910
00911     return ilo;
00912 }

```

```

00913
00914 /*****
00915
00916 int locate_reg(
00917     double **xx,
00918     int n,
00919     double x) {
00920
00921     /* Calculate index... */
00922     int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
00923
00924     /* Check range... */
00925     if (i < 0)
00926         i = 0;
00927     else if (i >= n - 2)
00928         i = n - 2;
00929
00930     return i;
00931 }
00932
00933 /*****
00934
00935 int read_atm(
00936     const char *filename,
00937     ctl_t * ctl,
00938     atm_t * atm) {
00939
00940     FILE *in;
00941
00942     char line[LEN], *tok;
00943
00944     double t0;
00945
00946     int dimid, ip, iq, ncid, varid;
00947
00948     size_t nparts;
00949
00950     /* Init... */
00951     atm->np = 0;
00952
00953     /* Write info... */
00954     printf("Read atmospheric data: %s\n", filename);
00955
00956     /* Read ASCII data... */
00957     if (ctl->atm_type == 0) {
00958
00959         /* Open file... */
00960         if (!(in = fopen(filename, "r"))) {
00961             WARN("File not found!");
00962             return 0;
00963         }
00964
00965         /* Read line... */
00966         while (fgets(line, LEN, in)) {
00967
00968             /* Read data... */
00969             TOK(line, tok, "%lg", atm->time[atm->np]);
00970             TOK(NULL, tok, "%lg", atm->p[atm->np]);
00971             TOK(NULL, tok, "%lg", atm->lon[atm->np]);
00972             TOK(NULL, tok, "%lg", atm->lat[atm->np]);
00973             for (iq = 0; iq < ctl->nq; iq++)
00974                 TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00975
00976             /* Convert altitude to pressure... */
00977             atm->p[atm->np] = P(atm->p[atm->np]);
00978
00979             /* Increment data point counter... */
00980             if ((++atm->np) > NP)
00981                 ERRMSG("Too many data points!");
00982         }
00983
00984         /* Close file... */
00985         fclose(in);
00986     }
00987
00988     /* Read binary data... */
00989     else if (ctl->atm_type == 1) {
00990
00991         /* Open file... */
00992         if (!(in = fopen(filename, "r")))
00993             return 0;
00994
00995         /* Read data... */
00996         FREAD(&atm->np, int, 1, in);
00997         FREAD(atm->time, double,
00998             (size_t) atm->np,
00999             in);

```

```

01000     FREAD(atm->p, double,
01001           (size_t) atm->np,
01002           in);
01003     FREAD(atm->lon, double,
01004           (size_t) atm->np,
01005           in);
01006     FREAD(atm->lat, double,
01007           (size_t) atm->np,
01008           in);
01009     for (iq = 0; iq < ctl->nq; iq++)
01010         FREAD(atm->q[iq], double,
01011               (size_t) atm->np,
01012               in);
01013
01014     /* Close file... */
01015     fclose(in);
01016 }
01017
01018 /* Read netCDF data... */
01019 else if (ctl->atm_type == 2) {
01020
01021     /* Open file... */
01022     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
01023         return 0;
01024
01025     /* Get dimensions... */
01026     NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
01027     NC(nc_inq_dimlen(ncid, dimid, &nparts));
01028     atm->np = (int) nparts;
01029     if (atm->np > NP)
01030         ERRMSG("Too many particles!");
01031
01032     /* Get time... */
01033     NC(nc_inq_varid(ncid, "time", &varid));
01034     NC(nc_get_var_double(ncid, varid, &t0));
01035     for (ip = 0; ip < atm->np; ip++)
01036         atm->time[ip] = t0;
01037
01038     /* Read geolocations... */
01039     NC(nc_inq_varid(ncid, "PRESS", &varid));
01040     NC(nc_get_var_double(ncid, varid, atm->p));
01041     NC(nc_inq_varid(ncid, "LON", &varid));
01042     NC(nc_get_var_double(ncid, varid, atm->lon));
01043     NC(nc_inq_varid(ncid, "LAT", &varid));
01044     NC(nc_get_var_double(ncid, varid, atm->lat));
01045
01046     /* Read variables... */
01047     if (ctl->qnt_p >= 0)
01048         if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
01049             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
01050     if (ctl->qnt_t >= 0)
01051         if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
01052             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
01053     if (ctl->qnt_u >= 0)
01054         if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
01055             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
01056     if (ctl->qnt_v >= 0)
01057         if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
01058             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
01059     if (ctl->qnt_w >= 0)
01060         if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
01061             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
01062     if (ctl->qnt_h2o >= 0)
01063         if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
01064             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
01065     if (ctl->qnt_o3 >= 0)
01066         if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
01067             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
01068     if (ctl->qnt_theta >= 0)
01069         if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
01070             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
01071     if (ctl->qnt_pv >= 0)
01072         if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
01073             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
01074
01075     /* Check data... */
01076     for (ip = 0; ip < atm->np; ip++)
01077         if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
01078             || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
01079             || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
01080             || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
01081             || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10) {
01082         atm->time[ip] = GSL_NAN;
01083         atm->p[ip] = GSL_NAN;
01084         atm->lon[ip] = GSL_NAN;
01085         atm->lat[ip] = GSL_NAN;
01086         for (iq = 0; iq < ctl->nq; iq++)

```

```

01087         atm->q[iq][ip] = GSL_NAN;
01088     } else {
01089         if (ctl->qnt_h2o >= 0)
01090             atm->q[ctl->qnt_h2o][ip] *= 1.608;
01091         if (ctl->qnt_pv >= 0)
01092             atm->q[ctl->qnt_pv][ip] *= 1e6;
01093         if (atm->lon[ip] > 180)
01094             atm->lon[ip] -= 360;
01095     }
01096
01097     /* Close file... */
01098     NC(nc_close(ncid));
01099 }
01100
01101 /* Error... */
01102 else
01103     ERRMSG("Atmospheric data type not supported!");
01104
01105 /* Check number of points... */
01106 if (atm->np < 1)
01107     ERRMSG("Can not read any data!");
01108
01109 /* Return success... */
01110 return 1;
01111 }
01112
01113 /*****
01114
01115 void read_ctl(
01116     const char *filename,
01117     int argc,
01118     char *argv[],
01119     ctl_t *ctl) {
01120
01121     /* Write info... */
01122     printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
01123           "(executable: %s | compiled: %s, %s)\n\n",
01124           argv[0], __DATE__, __TIME__);
01125
01126     /* Initialize quantity indices... */
01127     ctl->qnt_ens = -1;
01128     ctl->qnt_m = -1;
01129     ctl->qnt_r = -1;
01130     ctl->qnt_rho = -1;
01131     ctl->qnt_ps = -1;
01132     ctl->qnt_pt = -1;
01133     ctl->qnt_z = -1;
01134     ctl->qnt_p = -1;
01135     ctl->qnt_t = -1;
01136     ctl->qnt_u = -1;
01137     ctl->qnt_v = -1;
01138     ctl->qnt_w = -1;
01139     ctl->qnt_h2o = -1;
01140     ctl->qnt_o3 = -1;
01141     ctl->qnt_theta = -1;
01142     ctl->qnt_vh = -1;
01143     ctl->qnt_vz = -1;
01144     ctl->qnt_pv = -1;
01145     ctl->qnt_tice = -1;
01146     ctl->qnt_tsts = -1;
01147     ctl->qnt_tnat = -1;
01148     ctl->qnt_stat = -1;
01149
01150     /* Read quantities... */
01151     ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
01152     if (ctl->nq > NQ)
01153         ERRMSG("Too many quantities!");
01154     for (int iq = 0; iq < ctl->nq; iq++) {
01155
01156         /* Read quantity name and format... */
01157         scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
01158         scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
01159                 ctl->qnt_format[iq]);
01160
01161         /* Try to identify quantity... */
01162         if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
01163             ctl->qnt_ens = iq;
01164             sprintf(ctl->qnt_unit[iq], "-");
01165         } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
01166             ctl->qnt_m = iq;
01167             sprintf(ctl->qnt_unit[iq], "kg");
01168         } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
01169             ctl->qnt_r = iq;
01170             sprintf(ctl->qnt_unit[iq], "m");
01171         } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
01172             ctl->qnt_rho = iq;
01173             sprintf(ctl->qnt_unit[iq], "kg/m^3");

```

```

01174     } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
01175         ctl->qnt_ps = iq;
01176         sprintf(ctl->qnt_unit[iq], "hPa");
01177     } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
01178         ctl->qnt_pt = iq;
01179         sprintf(ctl->qnt_unit[iq], "hPa");
01180     } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
01181         ctl->qnt_z = iq;
01182         sprintf(ctl->qnt_unit[iq], "km");
01183     } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
01184         ctl->qnt_p = iq;
01185         sprintf(ctl->qnt_unit[iq], "hPa");
01186     } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
01187         ctl->qnt_t = iq;
01188         sprintf(ctl->qnt_unit[iq], "K");
01189     } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
01190         ctl->qnt_u = iq;
01191         sprintf(ctl->qnt_unit[iq], "m/s");
01192     } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
01193         ctl->qnt_v = iq;
01194         sprintf(ctl->qnt_unit[iq], "m/s");
01195     } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
01196         ctl->qnt_w = iq;
01197         sprintf(ctl->qnt_unit[iq], "hPa/s");
01198     } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
01199         ctl->qnt_h2o = iq;
01200         sprintf(ctl->qnt_unit[iq], "l");
01201     } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
01202         ctl->qnt_o3 = iq;
01203         sprintf(ctl->qnt_unit[iq], "l");
01204     } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
01205         ctl->qnt_theta = iq;
01206         sprintf(ctl->qnt_unit[iq], "K");
01207     } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
01208         ctl->qnt_vh = iq;
01209         sprintf(ctl->qnt_unit[iq], "m/s");
01210     } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
01211         ctl->qnt_vz = iq;
01212         sprintf(ctl->qnt_unit[iq], "m/s");
01213     } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
01214         ctl->qnt_pv = iq;
01215         sprintf(ctl->qnt_unit[iq], "PVU");
01216     } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
01217         ctl->qnt_tice = iq;
01218         sprintf(ctl->qnt_unit[iq], "K");
01219     } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
01220         ctl->qnt_tsts = iq;
01221         sprintf(ctl->qnt_unit[iq], "K");
01222     } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
01223         ctl->qnt_tnat = iq;
01224         sprintf(ctl->qnt_unit[iq], "K");
01225     } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
01226         ctl->qnt_stat = iq;
01227         sprintf(ctl->qnt_unit[iq], "-");
01228     } else
01229         scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
01230 }
01231
01232 /* Check quantity flags... */
01233 if (ctl->qnt_tsts >= 0)
01234     if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
01235         ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
01236
01237 /* Time steps of simulation... */
01238 ctl->direction =
01239     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
01240 if (ctl->direction != -1 && ctl->direction != 1)
01241     ERRMSG("Set DIRECTION to -1 or 1!");
01242 ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
01243 ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
01244
01245 /* Meteorological data... */
01246 ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
01247 ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
01248 ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
01249 ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
01250 ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
01251 ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
01252 ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
01253 ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
01254 if (ctl->met_np > EP)
01255     ERRMSG("Too many levels!");
01256 for (int ip = 0; ip < ctl->met_np; ip++)
01257     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
01258 ctl->met_tropo
01259     = (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "0", NULL);
01260 scan_ctl(filename, argc, argv, "MET_GEOPOT", -1, "-", ctl->met_geopot);

```

```

01261 scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
01262 ctl->met_dt_out =
01263     scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
01264
01265 /* Isosurface parameters... */
01266 ctl->isosurf
01267     = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
01268 scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
01269
01270 /* Diffusion parameters... */
01271 ctl->turb_dx_trop
01272     = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
01273 ctl->turb_dx_strat
01274     = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
01275 ctl->turb_dz_trop
01276     = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
01277 ctl->turb_dz_strat
01278     = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
01279 ctl->turb_mesox =
01280     scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
01281 ctl->turb_mesoz =
01282     scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
01283
01284 /* Mass and life time... */
01285 ctl->molmass = scan_ctl(filename, argc, argv, "MOLMASS", -1, "1", NULL);
01286 ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
01287 ctl->tdec_strat =
01288     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
01289
01290 /* PSC analysis... */
01291 ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
01292 ctl->psc_hno3 =
01293     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
01294
01295 /* Output of atmospheric data... */
01296 scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
atm_basename);
01297 scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
01298 ctl->atm_dt_out =
01299     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
01300 ctl->atm_filter =
01301     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
01302 ctl->atm_stride =
01303     (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
01304 ctl->atm_type =
01305     (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
01306
01307 /* Output of CSI data... */
01308 scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
csi_basename);
01309 ctl->csi_dt_out =
01310     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
01311 scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->
csi_obsfile);
01312 ctl->csi_obsmin =
01313     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
01314 ctl->csi_modmin =
01315     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
01316 ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
01317 ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
01318 ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
01319 ctl->csi_lon0 =
01320     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
01321 ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
01322 ctl->csi_nx =
01323     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
01324 ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
01325 ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
01326 ctl->csi_ny =
01327     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
01328
01329 /* Output of ensemble data... */
01330 scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
ens_basename);
01331
01332 /* Output of grid data... */
01333 scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
ctl->grid_basename);
01334 scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
grid_gpfile);
01335 ctl->grid_dt_out =
01336     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
01337 ctl->grid_sparse =
01338     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
01339 ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
01340 ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
01341 ctl->grid_nz =

```



```

01343     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
01344     ctl->grid_lon0 =
01345         scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
01346     ctl->grid_lon1 =
01347         scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
01348     ctl->grid_nx =
01349         (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
01350     ctl->grid_lat0 =
01351         scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
01352     ctl->grid_lat1 =
01353         scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
01354     ctl->grid_ny =
01355         (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
01356
01357     /* Output of profile data... */
01358     scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
01359         ctl->prof_basename);
01360     scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
prof_obsfile);
01361     ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
01362     ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
01363     ctl->prof_nz =
01364         (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
01365     ctl->prof_lon0 =
01366         scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
01367     ctl->prof_lon1 =
01368         scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
01369     ctl->prof_nx =
01370         (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
01371     ctl->prof_lat0 =
01372         scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
01373     ctl->prof_lat1 =
01374         scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
01375     ctl->prof_ny =
01376         (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
01377
01378     /* Output of station data... */
01379     scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
01380         ctl->stat_basename);
01381     ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
01382     ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
01383     ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
01384 }
01385
01386 /*****
01387
01388 int read_met(
01389     ctl_t * ctl,
01390     char *filename,
01391     met_t * met) {
01392
01393     char cmd[2 * LEN], levname[LEN], tstr[10];
01394
01395     float help[EX * EY];
01396
01397     int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
01398
01399     size_t np, nx, ny;
01400
01401     /* Write info... */
01402     printf("Read meteorological data: %s\n", filename);
01403
01404     /* Get time from filename... */
01405     sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
01406     year = atoi(tstr);
01407     sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
01408     mon = atoi(tstr);
01409     sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
01410     day = atoi(tstr);
01411     sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
01412     hour = atoi(tstr);
01413     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
01414
01415     /* Open netCDF file... */
01416     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
01417
01418         /* Try to stage meteo file... */
01419         if (ctl->met_stage[0] != '-') {
01420             sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
01421                 year, mon, day, hour, filename);
01422             if (system(cmd) != 0)
01423                 ERRMSG("Error while staging meteo data!");
01424         }
01425
01426         /* Try to open again... */
01427         if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
01428             WARN("File not found!");

```

```

01429         return 0;
01430     }
01431 }
01432
01433 /* Get dimensions... */
01434 NC(nc_inq_dimid(ncid, "lon", &dimid));
01435 NC(nc_inq_dimlen(ncid, dimid, &nx));
01436 if (nx < 2 || nx > EX)
01437     ERRMSG("Number of longitudes out of range!");
01438
01439 NC(nc_inq_dimid(ncid, "lat", &dimid));
01440 NC(nc_inq_dimlen(ncid, dimid, &ny));
01441 if (ny < 2 || ny > EY)
01442     ERRMSG("Number of latitudes out of range!");
01443
01444 sprintf(levname, "lev");
01445 NC(nc_inq_dimid(ncid, levname, &dimid));
01446 NC(nc_inq_dimlen(ncid, dimid, &np));
01447 if (np == 1) {
01448     sprintf(levname, "lev_2");
01449     NC(nc_inq_dimid(ncid, levname, &dimid));
01450     NC(nc_inq_dimlen(ncid, dimid, &np));
01451 }
01452 if (np < 2 || np > EP)
01453     ERRMSG("Number of levels out of range!");
01454
01455 /* Store dimensions... */
01456 met->np = (int) np;
01457 met->nx = (int) nx;
01458 met->ny = (int) ny;
01459
01460 /* Get horizontal grid... */
01461 NC(nc_inq_varid(ncid, "lon", &varid));
01462 NC(nc_get_var_double(ncid, varid, met->lon));
01463 NC(nc_inq_varid(ncid, "lat", &varid));
01464 NC(nc_get_var_double(ncid, varid, met->lat));
01465
01466 /* Read meteorological data... */
01467 read_met_help(ncid, "t", "T", met, met->t, 1.0);
01468 read_met_help(ncid, "u", "U", met, met->u, 1.0);
01469 read_met_help(ncid, "v", "V", met, met->v, 1.0);
01470 read_met_help(ncid, "w", "W", met, met->w, 0.01f);
01471 read_met_help(ncid, "q", "Q", met, met->h2o, (float) (MA / 18.01528));
01472 read_met_help(ncid, "o3", "O3", met, met->o3, (float) (MA / 48.00));
01473
01474 /* Meteo data on pressure levels... */
01475 if (ctl->met_np <= 0) {
01476
01477     /* Read pressure levels from file... */
01478     NC(nc_inq_varid(ncid, levname, &varid));
01479     NC(nc_get_var_double(ncid, varid, met->p));
01480     for (ip = 0; ip < met->np; ip++)
01481         met->p[ip] /= 100.;
01482
01483     /* Extrapolate data for lower boundary... */
01484     read_met_extrapolate(met);
01485 }
01486
01487 /* Meteo data on model levels... */
01488 else {
01489
01490     /* Read pressure data from file... */
01491     read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
01492
01493     /* Interpolate from model levels to pressure levels... */
01494     read_met_ml2pl(ctl, met, met->t);
01495     read_met_ml2pl(ctl, met, met->u);
01496     read_met_ml2pl(ctl, met, met->v);
01497     read_met_ml2pl(ctl, met, met->w);
01498     read_met_ml2pl(ctl, met, met->h2o);
01499     read_met_ml2pl(ctl, met, met->o3);
01500
01501     /* Set pressure levels... */
01502     met->np = ctl->met_np;
01503     for (ip = 0; ip < met->np; ip++)
01504         met->p[ip] = ctl->met_p[ip];
01505 }
01506
01507 /* Check ordering of pressure levels... */
01508 for (ip = 1; ip < met->np; ip++)
01509     if (met->p[ip - 1] < met->p[ip])
01510         ERRMSG("Pressure levels must be descending!");
01511
01512 /* Read surface pressure... */
01513 if (nc_inq_varid(ncid, "ps", &varid) == NC_NOERR
01514     || nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
01515     NC(nc_get_var_float(ncid, varid, help));

```

```

01516     for (iy = 0; iy < met->ny; iy++)
01517     for (ix = 0; ix < met->nx; ix++)
01518         met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
01519 } else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR
01520           || nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR) {
01521     NC(nc_get_var_float(ncid, varid, help));
01522     for (iy = 0; iy < met->ny; iy++)
01523     for (ix = 0; ix < met->nx; ix++)
01524         met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
01525 } else
01526     for (ix = 0; ix < met->nx; ix++)
01527     for (iy = 0; iy < met->ny; iy++)
01528         met->ps[ix][iy] = met->p[0];
01529
01530 /* Create periodic boundary conditions... */
01531 read_met_periodic(met);
01532
01533 /* Calculate geopotential heights... */
01534 read_met_geopot(ctl, met);
01535
01536 /* Calculate potential vorticity... */
01537 read_met_pv(met);
01538
01539 /* Calculate tropopause pressure... */
01540 read_met_tropo(ctl, met);
01541
01542 /* Downsampling... */
01543 read_met_sample(ctl, met);
01544
01545 /* Close file... */
01546 NC(nc_close(ncid));
01547
01548 /* Return success... */
01549 return 1;
01550 }
01551
01552 /*****
01553 void read_met_extrapolate(
01554     met_t * met) {
01555     int ip, ip0, ix, iy;
01556
01557     /* Loop over columns... */
01558 #pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
01559     for (ix = 0; ix < met->nx; ix++)
01560     for (iy = 0; iy < met->ny; iy++) {
01561
01562         /* Find lowest valid data point... */
01563         for (ip0 = met->np - 1; ip0 >= 0; ip0--)
01564             if (!gsl_finite(met->t[ix][iy][ip0])
01565                 || !gsl_finite(met->u[ix][iy][ip0])
01566                 || !gsl_finite(met->v[ix][iy][ip0])
01567                 || !gsl_finite(met->w[ix][iy][ip0]))
01568                 break;
01569
01570         /* Extrapolate... */
01571         for (ip = ip0; ip >= 0; ip--) {
01572             met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
01573             met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
01574             met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
01575             met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
01576             met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
01577             met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
01578         }
01579     }
01580 }
01581 }
01582
01583 /*****
01584 void read_met_geopot(
01585     ctl_t * ctl,
01586     met_t * met) {
01587     const int dx = 6, dy = 4;
01588
01589     static double logp[EP], topo_lat[EY], topo_lon[EX], topo_z[EX][EY];
01590     static float help[EX][EY][EP];
01591
01592     static int init, topo_nx = -1, topo_ny;
01593
01594     FILE *in;
01595
01596     char line[LEN];
01597
01598     double lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;

```

```

01603
01604     int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;
01605
01606     /* Initialize geopotential heights... */
01607 #pragma omp parallel for default(shared) private(ix,iy,ip)
01608     for (ix = 0; ix < met->nx; ix++)
01609         for (iy = 0; iy < met->ny; iy++)
01610             for (ip = 0; ip < met->np; ip++)
01611                 met->z[ix][iy][ip] = GSL_NAN;
01612
01613     /* Check filename... */
01614     if (ctl->met_geopot[0] == '-')
01615         return;
01616
01617     /* Read surface geopotential... */
01618     if (!init) {
01619         init = 1;
01620
01621         /* Write info... */
01622         printf("Read surface geopotential: %s\n", ctl->met_geopot);
01623
01624         /* Open file... */
01625         if (!(in = fopen(ctl->met_geopot, "r")))
01626             ERRMSG("Cannot open file!");
01627
01628         /* Read data... */
01629         while (fgets(line, LEN, in))
01630             if (sscanf(line, "%lg %lg %lg", &r lon, &r lat, &r z) == 3) {
01631                 if (r lon != r lon_old) {
01632                     if ((++topo_nx) > EX)
01633                         ERRMSG("Too many longitudes!");
01634                     topo_ny = 0;
01635                 }
01636                 r lon_old = r lon;
01637                 topo_lon[topo_nx] = r lon;
01638                 topo_lat[topo_ny] = r lat;
01639                 topo_z[topo_nx][topo_ny] = r z;
01640                 if ((++topo_ny) > EY)
01641                     ERRMSG("Too many latitudes!");
01642             }
01643         if ((++topo_nx) > EX)
01644             ERRMSG("Too many longitudes!");
01645
01646         /* Close file... */
01647         fclose(in);
01648
01649         /* Check grid spacing... */
01650         if (fabs(met->lon[0] - met->lon[1]) != fabs(topo_lon[0] - topo_lon[1])
01651             || fabs(met->lat[0] - met->lat[1]) != fabs(topo_lat[0] - topo_lat[1]))
01652             WARN("Grid spacing does not match!");
01653
01654         /* Calculate log pressure... */
01655         for (ip = 0; ip < met->np; ip++)
01656             logp[ip] = log(met->p[ip]);
01657     }
01658
01659     /* Apply hydrostatic equation to calculate geopotential heights... */
01660 #pragma omp parallel for default(shared) private(ix,iy,lon,lat,tx,ty,z0,z1,ip0,ts,ip)
01661     for (ix = 0; ix < met->nx; ix++) {
01662
01663         /* Get longitude index... */
01664         lon = met->lon[ix];
01665         if (lon < topo_lon[0])
01666             lon += 360;
01667         else if (lon > topo_lon[topo_nx - 1])
01668             lon -= 360;
01669         tx = locate_reg(topo_lon, topo_nx, lon);
01670
01671         /* Loop over latitudes... */
01672         for (iy = 0; iy < met->ny; iy++) {
01673
01674             /* Get latitude index... */
01675             lat = met->lat[iy];
01676             ty = locate_reg(topo_lat, topo_ny, lat);
01677
01678             /* Get surface height... */
01679             z0 = LIN(topo_lon[tx], topo_z[tx][ty],
01680                     topo_lon[tx + 1], topo_z[tx + 1][ty], lon);
01681             z1 = LIN(topo_lon[tx], topo_z[tx][ty + 1],
01682                     topo_lon[tx + 1], topo_z[tx + 1][ty + 1], lon);
01683             z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);
01684
01685             /* Find surface pressure level... */
01686             ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
01687
01688             /* Get surface data... */
01689             ts =

```

```

01690     LIN(met->p[ip0],
01691         TVIRT(met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]),
01692         met->p[ip0 + 1],
01693         TVIRT(met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]),
01694         met->ps[ix][iy]);
01695
01696     /* Upper part of profile... */
01697     met->z[ix][iy][ip0 + 1]
01698     = (float) (z0 + RI / MA / G0 * 0.5
01699               * (ts + TVIRT(met->t[ix][iy][ip0 + 1],
01700                             met->h2o[ix][iy][ip0 + 1]))
01701               * (log(met->ps[ix][iy]) - logp[ip0 + 1]));
01702     for (ip = ip0 + 2; ip < met->np; ip++)
01703         met->z[ix][iy][ip]
01704         = (float) (met->z[ix][iy][ip - 1] + RI / MA / G0 * 0.5 *
01705                   (TVIRT(met->t[ix][iy][ip - 1], met->h2o[ix][iy][ip - 1])
01706                     + TVIRT(met->t[ix][iy][ip], met->h2o[ix][iy][ip]))
01707                   * (logp[ip - 1] - logp[ip]));
01708 }
01709 }
01710
01711 /* Smoothing... */
01712 #pragma omp parallel for default(shared) private(ix,iy,ip,n,ix2,ix3,iy2)
01713 for (ix = 0; ix < met->nx; ix++)
01714     for (iy = 0; iy < met->ny; iy++)
01715         for (ip = 0; ip < met->np; ip++) {
01716             n = 0;
01717             help[ix][iy][ip] = 0;
01718             for (ix2 = ix - dx; ix2 <= ix + dx; ix2++) {
01719                 ix3 = ix2;
01720                 if (ix3 < 0)
01721                     ix3 += met->nx;
01722                 else if (ix3 >= met->nx)
01723                     ix3 -= met->nx;
01724                 for (iy2 = GSL_MAX(iy - dy, 0);
01725                     iy2 <= GSL_MIN(iy + dy, met->ny - 1); iy2++)
01726                     if (gsl_finite(met->z[ix3][iy2][ip])) {
01727                         help[ix][iy][ip] += met->z[ix3][iy2][ip];
01728                         n++;
01729                     }
01730             }
01731             if (n > 0)
01732                 help[ix][iy][ip] /= (float) n;
01733             else
01734                 help[ix][iy][ip] = GSL_NAN;
01735         }
01736
01737 /* Copy data... */
01738 #pragma omp parallel for default(shared) private(ix,iy,ip)
01739 for (ix = 0; ix < met->nx; ix++)
01740     for (iy = 0; iy < met->ny; iy++)
01741         for (ip = 0; ip < met->np; ip++)
01742             met->z[ix][iy][ip] = help[ix][iy][ip];
01743 }
01744
01745 /*****
01746
01747 void read_met_help(
01748     int ncid,
01749     char *varname,
01750     char *varname2,
01751     met_t *met,
01752     float dest[EX][EY][EP],
01753     float scl) {
01754
01755     float *help;
01756
01757     int ip, ix, iy, varid;
01758
01759     /* Check if variable exists... */
01760     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
01761         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
01762             return;
01763
01764     /* Allocate... */
01765     ALLOC(help, float, EX * EY * EP);
01766
01767     /* Read data... */
01768     NC(nc_get_var_float(ncid, varid, help));
01769
01770     /* Copy and check data... */
01771     #pragma omp parallel for default(shared) private(ix,iy,ip)
01772     for (ix = 0; ix < met->nx; ix++)
01773         for (iy = 0; iy < met->ny; iy++)
01774             for (ip = 0; ip < met->np; ip++) {
01775                 dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
01776                 if (fabsf(dest[ix][iy][ip]) < 1e14f)

```

```

01777         dest[ix][iy][ip] *= scl;
01778     else
01779         dest[ix][iy][ip] = GSL_NAN;
01780     }
01781 }
01782 /* Free... */
01783 free(help);
01784 }
01785
01786 /*****
01787
01788 void read_met_m12pl(
01789     ctl_t * ctl,
01790     met_t * met,
01791     float var[EX][EY][EP]) {
01792
01793     double aux[EP], p[EP], pt;
01794
01795     int ip, ip2, ix, iy;
01796
01797     /* Loop over columns... */
01798 #pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
01799     for (ix = 0; ix < met->nx; ix++)
01800         for (iy = 0; iy < met->ny; iy++) {
01801
01802             /* Copy pressure profile... */
01803             for (ip = 0; ip < met->np; ip++)
01804                 p[ip] = met->pl[ix][iy][ip];
01805
01806             /* Interpolate... */
01807             for (ip = 0; ip < ctl->met_np; ip++) {
01808                 pt = ctl->met_p[ip];
01809                 if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
01810                     pt = p[0];
01811                 else if ((pt > p[met->np - 1] && p[1] > p[0])
01812                        || (pt < p[met->np - 1] && p[1] < p[0]))
01813                     pt = p[met->np - 1];
01814                 ip2 = locate_irr(p, met->np, pt);
01815                 aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
01816                             p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
01817             }
01818
01819             /* Copy data... */
01820             for (ip = 0; ip < ctl->met_np; ip++)
01821                 var[ix][iy][ip] = (float) aux[ip];
01822         }
01823     }
01824
01825 /*****
01826
01827 void read_met_periodic(
01828     met_t * met) {
01829
01830     /* Check longitudes... */
01831     if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
01832              + met->lon[1] - met->lon[0] - 360) < 0.01))
01833         return;
01834
01835     /* Increase longitude counter... */
01836     if ((++met->nx) > EX)
01837         ERRMSG("Cannot create periodic boundary conditions!");
01838
01839     /* Set longitude... */
01840     met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
lon[0];
01841
01842     /* Loop over latitudes and pressure levels... */
01843 #pragma omp parallel for default(shared)
01844     for (int iy = 0; iy < met->ny; iy++) {
01845         met->ps[met->nx - 1][iy] = met->ps[0][iy];
01846         met->pt[met->nx - 1][iy] = met->pt[0][iy];
01847         for (int ip = 0; ip < met->np; ip++) {
01848             met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
01849             met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
01850             met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
01851             met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
01852             met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
01853             met->pv[met->nx - 1][iy][ip] = met->pv[0][iy][ip];
01854             met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
01855             met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01856         }
01857     }
01858 }
01859
01860 /*****
01861
01862 void read_met_pv(

```

```

01863     met_t * met) {
01864
01865     double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
01866           dtdp, dudp, dvdp, latr, vort, pows[EP];
01867
01868     int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
01869
01870     /* Set powers... */
01871     for (ip = 0; ip < met->np; ip++)
01872         pows[ip] = pow(1000. / met->p[ip], 0.286);
01873
01874     /* Loop over grid points... */
01875     #pragma omp parallel for default(shared)
01876     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)
01877     for (ix = 0; ix < met->nx; ix++) {
01878
01879         /* Set indices... */
01880         ix0 = GSL_MAX(ix - 1, 0);
01881         ix1 = GSL_MIN(ix + 1, met->nx - 1);
01882
01883         /* Loop over grid points... */
01884         for (iy = 0; iy < met->ny; iy++) {
01885
01886             /* Set indices... */
01887             iy0 = GSL_MAX(iy - 1, 0);
01888             iy1 = GSL_MIN(iy + 1, met->ny - 1);
01889
01890             /* Set auxiliary variables... */
01891             latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
01892             dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
01893             dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
01894             c0 = cos(met->lat[iy0] / 180. * M_PI);
01895             c1 = cos(met->lat[iy1] / 180. * M_PI);
01896             cr = cos(latr / 180. * M_PI);
01897             vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
01898
01899             /* Loop over grid points... */
01900             for (ip = 0; ip < met->np; ip++) {
01901
01902                 /* Get gradients in longitude... */
01903                 dtdx = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
01904                 dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
01905
01906                 /* Get gradients in latitude... */
01907                 dtdy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
01908                 dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
01909
01910                 /* Set indices... */
01911                 ip0 = GSL_MAX(ip - 1, 0);
01912                 ip1 = GSL_MIN(ip + 1, met->np - 1);
01913
01914                 /* Get gradients in pressure... */
01915                 dp0 = 100. * (met->p[ip] - met->p[ip0]);
01916                 dp1 = 100. * (met->p[ip1] - met->p[ip]);
01917                 if (ip != ip0 && ip != ip1) {
01918                     denom = dp0 * dp1 * (dp0 + dp1);
01919                     dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
01920                           - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
01921                           + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
01922                           / denom;
01923                     dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
01924                           - dp1 * dp1 * met->u[ix][iy][ip0]
01925                           + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
01926                           / denom;
01927                     dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
01928                           - dp1 * dp1 * met->v[ix][iy][ip0]
01929                           + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
01930                           / denom;
01931                 } else {
01932                     denom = dp0 + dp1;
01933                     dtdp =
01934                         (met->t[ix][iy][ip1] * pows[ip1] -
01935                         met->t[ix][iy][ip0] * pows[ip0]) / denom;
01936                     dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
01937                     dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
01938                 }
01939
01940                 /* Calculate PV... */
01941                 met->pv[ix][iy][ip] = (float)
01942                     (1e6 * G0 *
01943                     (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
01944             }
01945         }
01946     }
01947     /* Fix for polar regions... */
01948     #pragma omp parallel for default(shared) private(ix,ip)

```

```

01949     for (ix = 0; ix < met->nx; ix++)
01950         for (ip = 0; ip < met->np; ip++) {
01951             met->pv[ix][0][ip]
01952                 = met->pv[ix][1][ip]
01953                 = met->pv[ix][2][ip];
01954             met->pv[ix][met->ny - 1][ip]
01955                 = met->pv[ix][met->ny - 2][ip]
01956                 = met->pv[ix][met->ny - 3][ip];
01957         }
01958     }
01959
01960     /*****
01961
01962 void read_met_sample(
01963     ctl_t * ctl,
01964     met_t * met) {
01965
01966     met_t *help;
01967
01968     float w, wsum;
01969
01970     int ip, ip2, ix, ix2, ix3, iy, iy2;
01971
01972     /* Check parameters... */
01973     if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
01974         && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
01975         return;
01976
01977     /* Allocate... */
01978     ALLOC(help, met_t, 1);
01979
01980     /* Copy data... */
01981     help->nx = met->nx;
01982     help->ny = met->ny;
01983     help->np = met->np;
01984     memcpy(help->lon, met->lon, sizeof(met->lon));
01985     memcpy(help->lat, met->lat, sizeof(met->lat));
01986     memcpy(help->p, met->p, sizeof(met->p));
01987
01988     /* Smoothing... */
01989     for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
01990         for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
01991             for (ip = 0; ip < met->np; ip += ctl->met_dp) {
01992                 help->ps[ix][iy] = 0;
01993                 help->pt[ix][iy] = 0;
01994                 help->z[ix][iy][ip] = 0;
01995                 help->t[ix][iy][ip] = 0;
01996                 help->u[ix][iy][ip] = 0;
01997                 help->v[ix][iy][ip] = 0;
01998                 help->w[ix][iy][ip] = 0;
01999                 help->pv[ix][iy][ip] = 0;
02000                 help->h2o[ix][iy][ip] = 0;
02001                 help->o3[ix][iy][ip] = 0;
02002                 wsum = 0;
02003                 for (ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1; ix2++) {
02004                     ix3 = ix2;
02005                     if (ix3 < 0)
02006                         ix3 += met->nx;
02007                     else if (ix3 >= met->nx)
02008                         ix3 -= met->nx;
02009
02010                     for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
02011                         iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
02012                         for (ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
02013                             ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
02014                             w = (float) (1.0 - fabs(ix - ix2) / ctl->met_sx)
02015                                 * (float) (1.0 - fabs(iy - iy2) / ctl->met_sy)
02016                                 * (float) (1.0 - fabs(ip - ip2) / ctl->met_sp);
02017                             help->ps[ix][iy] += w * met->ps[ix3][iy2];
02018                             help->pt[ix][iy] += w * met->pt[ix3][iy2];
02019                             help->z[ix][iy][ip] += w * met->z[ix3][iy2][ip2];
02020                             help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
02021                             help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
02022                             help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
02023                             help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
02024                             help->pv[ix][iy][ip] += w * met->pv[ix3][iy2][ip2];
02025                             help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
02026                             help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
02027                             wsum += w;
02028                         }
02029                     }
02030                 help->ps[ix][iy] /= wsum;
02031                 help->pt[ix][iy] /= wsum;
02032                 help->t[ix][iy][ip] /= wsum;
02033                 help->z[ix][iy][ip] /= wsum;
02034                 help->u[ix][iy][ip] /= wsum;
02035                 help->v[ix][iy][ip] /= wsum;

```



```

02036         help->w[ix][iy][ip] /= wsum;
02037         help->pv[ix][iy][ip] /= wsum;
02038         help->h2o[ix][iy][ip] /= wsum;
02039         help->o3[ix][iy][ip] /= wsum;
02040     }
02041 }
02042 }
02043
02044 /* Downsampling... */
02045 met->nx = 0;
02046 for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
02047     met->lon[met->nx] = help->lon[ix];
02048     met->ny = 0;
02049     for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
02050         met->lat[met->ny] = help->lat[iy];
02051         met->ps[met->nx][met->ny] = help->ps[ix][iy];
02052         met->pt[met->nx][met->ny] = help->pt[ix][iy];
02053         met->np = 0;
02054         for (ip = 0; ip < help->np; ip += ctl->met_dp) {
02055             met->p[met->np] = help->p[ip];
02056             met->z[met->nx][met->ny][met->np] = help->z[ix][iy][ip];
02057             met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
02058             met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
02059             met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
02060             met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
02061             met->pv[met->nx][met->ny][met->np] = help->pv[ix][iy][ip];
02062             met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
02063             met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
02064             met->np++;
02065         }
02066         met->ny++;
02067     }
02068     met->nx++;
02069 }
02070
02071 /* Free... */
02072 free(help);
02073 }
02074
02075 /*****
02076
02077 void read_met_tropo(
02078     ctl_t * ctl,
02079     met_t * met) {
02080
02081     double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
02082         th2[200], z[EP], z2[200];
02083
02084     int found, ix, iy, iz, iz2;
02085
02086     /* Get altitude and pressure profiles... */
02087     for (iz = 0; iz < met->np; iz++)
02088         z[iz] = Z(met->p[iz]);
02089     for (iz = 0; iz <= 170; iz++) {
02090         z2[iz] = 4.5 + 0.1 * iz;
02091         p2[iz] = P(z2[iz]);
02092     }
02093
02094     /* Do not calculate tropopause... */
02095     if (ctl->met_tropo == 0)
02096         for (ix = 0; ix < met->nx; ix++)
02097             for (iy = 0; iy < met->ny; iy++)
02098                 met->pt[ix][iy] = GSL_NAN;
02099
02100     /* Use tropopause climatology... */
02101     else if (ctl->met_tropo == 1) {
02102 #pragma omp parallel for default(shared) private(ix,iy)
02103         for (ix = 0; ix < met->nx; ix++)
02104             for (iy = 0; iy < met->ny; iy++)
02105                 met->pt[ix][iy] = clim_tropo(met->time, met->lat[iy]);
02106     }
02107
02108     /* Use cold point... */
02109     else if (ctl->met_tropo == 2) {
02110
02111         /* Loop over grid points... */
02112 #pragma omp parallel for default(shared) private(ix,iy,iz,t,t2)
02113         for (ix = 0; ix < met->nx; ix++)
02114             for (iy = 0; iy < met->ny; iy++) {
02115
02116                 /* Interpolate temperature profile... */
02117                 for (iz = 0; iz < met->np; iz++)
02118                     t[iz] = met->t[ix][iy][iz];
02119                 spline(z, t, met->np, z2, t2, 171);
02120
02121                 /* Find minimum... */
02122                 iz = (int) gsl_stats_min_index(t2, 1, 171);

```

```

02123         if (iz <= 0 || iz >= 170)
02124             met->pt[ix][iy] = GSL_NAN;
02125         else
02126             met->pt[ix][iy] = p2[iz];
02127     }
02128 }
02129
02130 /* Use WMO definition... */
02131 else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
02132
02133     /* Loop over grid points... */
02134     #pragma omp parallel for default(shared) private(ix,iy,iz,iz2,t,t2,found)
02135     for (ix = 0; ix < met->nx; ix++)
02136         for (iy = 0; iy < met->ny; iy++) {
02137
02138             /* Interpolate temperature profile... */
02139             for (iz = 0; iz < met->np; iz++)
02140                 t[iz] = met->t[ix][iy][iz];
02141             spline(z, t, met->np, z2, t2, 161);
02142
02143             /* Find 1st tropopause... */
02144             met->pt[ix][iy] = GSL_NAN;
02145             for (iz = 0; iz <= 140; iz++) {
02146                 found = 1;
02147                 for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
02148                     if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
02149                         * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
02150                         found = 0;
02151                         break;
02152                     }
02153                 if (found) {
02154                     if (iz > 0 && iz < 140)
02155                         met->pt[ix][iy] = p2[iz];
02156                     break;
02157                 }
02158             }
02159
02160             /* Find 2nd tropopause... */
02161             if (ctl->met_tropo == 4) {
02162                 met->pt[ix][iy] = GSL_NAN;
02163                 for (; iz <= 140; iz++) {
02164                     found = 1;
02165                     for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
02166                         if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
02167                             * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) < 3.0) {
02168                             found = 0;
02169                             break;
02170                         }
02171                     if (found)
02172                         break;
02173                 }
02174                 for (; iz <= 140; iz++) {
02175                     found = 1;
02176                     for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
02177                         if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
02178                             * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
02179                             found = 0;
02180                             break;
02181                         }
02182                     if (found) {
02183                         if (iz > 0 && iz < 140)
02184                             met->pt[ix][iy] = p2[iz];
02185                         break;
02186                     }
02187                 }
02188             }
02189         }
02190     }
02191
02192     /* Use dynamical tropopause... */
02193     else if (ctl->met_tropo == 5) {
02194
02195         /* Loop over grid points... */
02196         #pragma omp parallel for default(shared) private(ix,iy,iz,pv,pv2,th,th2)
02197         for (ix = 0; ix < met->nx; ix++)
02198             for (iy = 0; iy < met->ny; iy++) {
02199
02200                 /* Interpolate potential vorticity profile... */
02201                 for (iz = 0; iz < met->np; iz++)
02202                     pv[iz] = met->pv[ix][iy][iz];
02203                 spline(z, pv, met->np, z2, pv2, 161);
02204
02205                 /* Interpolate potential temperature profile... */
02206                 for (iz = 0; iz < met->np; iz++)
02207                     th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
02208                 spline(z, th, met->np, z2, th2, 161);
02209

```

```

02210      /* Find dynamical tropopause 3.5 PVU + 380 K */
02211      met->pt[ix][iy] = GSL_NAN;
02212      for (iz = 0; iz <= 160; iz++)
02213          if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
02214              if (iz > 0 && iz < 160)
02215                  met->pt[ix][iy] = p2[iz];
02216              break;
02217          }
02218      }
02219  }
02220
02221  else
02222      ERRMSG("Cannot calculate tropopause!");
02223 }
02224
02225 /*****
02226
02227 double scan_ctl(
02228     const char *filename,
02229     int argc,
02230     char *argv[],
02231     const char *varname,
02232     int arridx,
02233     const char *defvalue,
02234     char *value) {
02235
02236     FILE *in = NULL;
02237
02238     char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
02239           msg[2 * LEN], rvarname[LEN], rval[LEN];
02240
02241     int contain = 0, i;
02242
02243     /* Open file... */
02244     if (filename[strlen(filename) - 1] != '-')
02245         if (!(in = fopen(filename, "r")))
02246             ERRMSG("Cannot open file!");
02247
02248     /* Set full variable name... */
02249     if (arridx >= 0) {
02250         sprintf(fullname1, "%s[%d]", varname, arridx);
02251         sprintf(fullname2, "%s[*]", varname);
02252     } else {
02253         sprintf(fullname1, "%s", varname);
02254         sprintf(fullname2, "%s", varname);
02255     }
02256
02257     /* Read data... */
02258     if (in != NULL)
02259         while (fgets(line, LEN, in))
02260             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
02261                 if (strcmp(rvarname, fullname1) == 0 ||
02262                     strcmp(rvarname, fullname2) == 0) {
02263                     contain = 1;
02264                     break;
02265                 }
02266     for (i = 1; i < argc - 1; i++)
02267         if (strcmp(argv[i], fullname1) == 0 ||
02268             strcmp(argv[i], fullname2) == 0) {
02269             sprintf(rval, "%s", argv[i + 1]);
02270             contain = 1;
02271             break;
02272         }
02273
02274     /* Close file... */
02275     if (in != NULL)
02276         fclose(in);
02277
02278     /* Check for missing variables... */
02279     if (!contain) {
02280         if (strlen(defvalue) > 0)
02281             sprintf(rval, "%s", defvalue);
02282         else {
02283             sprintf(msg, "Missing variable %s!\n", fullname1);
02284             ERRMSG(msg);
02285         }
02286     }
02287
02288     /* Write info... */
02289     printf("%s = %s\n", fullname1, rval);
02290
02291     /* Return values... */
02292     if (value != NULL)
02293         sprintf(value, "%s", rval);
02294     return atof(rval);
02295 }
02296

```

```

02297 /*****
02298
02299 void spline(
02300     double *x,
02301     double *y,
02302     int n,
02303     double *x2,
02304     double *y2,
02305     int n2) {
02306
02307     gsl_interp_accel *acc;
02308
02309     gsl_spline *s;
02310
02311     /* Allocate... */
02312     acc = gsl_interp_accel_alloc();
02313     s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
02314
02315     /* Interpolate temperature profile... */
02316     gsl_spline_init(s, x, y, (size_t) n);
02317     for (int i = 0; i < n2; i++)
02318         y2[i] = gsl_spline_eval(s, x2[i], acc);
02319
02320     /* Free... */
02321     gsl_spline_free(s);
02322     gsl_interp_accel_free(acc);
02323 }
02324
02325 /*****
02326
02327 double stddev(
02328     double *data,
02329     int n) {
02330
02331     if (n <= 0)
02332         return 0;
02333
02334     double avg = 0, rms = 0;
02335
02336     for (int i = 0; i < n; ++i)
02337         avg += data[i];
02338     avg /= n;
02339
02340     for (int i = 0; i < n; ++i)
02341         rms += SQR(data[i] - avg);
02342
02343     return sqrt(rms / (n - 1));
02344 }
02345
02346 /*****
02347
02348 void time2jsec(
02349     int year,
02350     int mon,
02351     int day,
02352     int hour,
02353     int min,
02354     int sec,
02355     double remain,
02356     double *jsec) {
02357
02358     struct tm t0, t1;
02359
02360     t0.tm_year = 100;
02361     t0.tm_mon = 0;
02362     t0.tm_mday = 1;
02363     t0.tm_hour = 0;
02364     t0.tm_min = 0;
02365     t0.tm_sec = 0;
02366
02367     t1.tm_year = year - 1900;
02368     t1.tm_mon = mon - 1;
02369     t1.tm_mday = day;
02370     t1.tm_hour = hour;
02371     t1.tm_min = min;
02372     t1.tm_sec = sec;
02373
02374     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
02375 }
02376
02377 /*****
02378
02379 void timer(
02380     const char *name,
02381     int id,
02382     int mode) {
02383

```

```

02384 static double starttime[NTIMER], runtime[NTIMER];
02385
02386 /* Check id... */
02387 if (id < 0 || id >= NTIMER)
02388     ERRMSG("Too many timers!");
02389
02390 /* Start timer... */
02391 if (mode == 1) {
02392     if (starttime[id] <= 0)
02393         starttime[id] = omp_get_wtime();
02394     else
02395         ERRMSG("Timer already started!");
02396 }
02397
02398 /* Stop timer... */
02399 else if (mode == 2) {
02400     if (starttime[id] > 0) {
02401         runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
02402         starttime[id] = -1;
02403     }
02404 }
02405
02406 /* Print timer... */
02407 else if (mode == 3) {
02408     printf("%s = %.3f s\n", name, runtime[id]);
02409     runtime[id] = 0;
02410 }
02411 }
02412
02413 /*****
02414
02415 void write_atm(
02416     const char *filename,
02417     ctl_t * ctl,
02418     atm_t * atm,
02419     double t) {
02420
02421     FILE *in, *out;
02422
02423     char line[LEN];
02424
02425     double r, t0, t1;
02426
02427     int ip, iq, year, mon, day, hour, min, sec;
02428
02429     /* Set time interval for output... */
02430     t0 = t - 0.5 * ctl->dt_mod;
02431     t1 = t + 0.5 * ctl->dt_mod;
02432
02433     /* Write info... */
02434     printf("Write atmospheric data: %s\n", filename);
02435
02436     /* Write ASCII data... */
02437     if (ctl->atm_type == 0) {
02438
02439         /* Check if gnuplot output is requested... */
02440         if (ctl->atm_gpfile[0] != '-') {
02441
02442             /* Create gnuplot pipe... */
02443             if (!(out = popen("gnuplot", "w")))
02444                 ERRMSG("Cannot create pipe to gnuplot!");
02445
02446             /* Set plot filename... */
02447             fprintf(out, "set out \"%s.png\"\n", filename);
02448
02449             /* Set time string... */
02450             jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02451             fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02452                 year, mon, day, hour, min);
02453
02454             /* Dump gnuplot file to pipe... */
02455             if (!(in = fopen(ctl->atm_gpfile, "r")))
02456                 ERRMSG("Cannot open file!");
02457             while (fgets(line, LEN, in))
02458                 fprintf(out, "%s", line);
02459             fclose(in);
02460         }
02461     }
02462     else {
02463
02464         /* Create file... */
02465         if (!(out = fopen(filename, "w")))
02466             ERRMSG("Cannot create file!");
02467     }
02468
02469     /* Write header... */
02470     fprintf(out,

```

```

02471         "# $1 = time [s]\n"
02472         "# $2 = altitude [km]\n"
02473         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02474     for (iq = 0; iq < ctl->nq; iq++)
02475         fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
02476                 ctl->qnt_unit[iq]);
02477     fprintf(out, "\n");
02478
02479     /* Write data... */
02480     for (ip = 0; ip < atm->np; ip += ctl->atm_stride) {
02481
02482         /* Check time... */
02483         if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
02484             continue;
02485
02486         /* Write output... */
02487         fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
02488                 atm->lon[ip], atm->lat[ip]);
02489         for (iq = 0; iq < ctl->nq; iq++) {
02490             fprintf(out, " ");
02491             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02492         }
02493         fprintf(out, "\n");
02494     }
02495
02496     /* Close file... */
02497     fclose(out);
02498 }
02499
02500 /* Write binary data... */
02501 else if (ctl->atm_type == 1) {
02502
02503     /* Create file... */
02504     if (!(out = fopen(filename, "w")))
02505         ERRMSG("Cannot create file!");
02506
02507     /* Write data... */
02508     FWRITE(&atm->np, int,
02509           1,
02510           out);
02511     FWRITE(atm->time, double,
02512           (size_t) atm->np,
02513           out);
02514     FWRITE(atm->p, double,
02515           (size_t) atm->np,
02516           out);
02517     FWRITE(atm->lon, double,
02518           (size_t) atm->np,
02519           out);
02520     FWRITE(atm->lat, double,
02521           (size_t) atm->np,
02522           out);
02523     for (iq = 0; iq < ctl->nq; iq++)
02524         FWRITE(atm->q[iq], double,
02525               (size_t) atm->np,
02526               out);
02527
02528     /* Close file... */
02529     fclose(out);
02530 }
02531
02532 /* Error... */
02533 else
02534     ERRMSG("Atmospheric data type not supported!");
02535 }
02536
02537 /*****
02538
02539 void write_csi(
02540     const char *filename,
02541     ctl_t * ctl,
02542     atm_t * atm,
02543     double t) {
02544
02545     static FILE *in, *out;
02546
02547     static char line[LEN];
02548
02549     static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
02550         rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
02551
02552     static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
02553
02554     /* Init... */
02555     if (t == ctl->t_start) {
02556
02557         /* Check quantity index for mass... */

```

```

02558     if (ctl->qnt_m < 0)
02559         ERRMSG("Need quantity mass!");
02560
02561     /* Open observation data file... */
02562     printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
02563     if (!(in = fopen(ctl->csi_obsfile, "r")))
02564         ERRMSG("Cannot open file!");
02565
02566     /* Create new file... */
02567     printf("Write CSI data: %s\n", filename);
02568     if !(out = fopen(filename, "w"))
02569         ERRMSG("Cannot create file!");
02570
02571     /* Write header... */
02572     fprintf(out,
02573         "# $1 = time [s]\n"
02574         "# $2 = number of hits (cx)\n"
02575         "# $3 = number of misses (cy)\n"
02576         "# $4 = number of false alarms (cz)\n"
02577         "# $5 = number of observations (cx + cy)\n"
02578         "# $6 = number of forecasts (cx + cz)\n"
02579         "# $7 = bias (forecasts/observations) [%%]\n"
02580         "# $8 = probability of detection (POD) [%%]\n"
02581         "# $9 = false alarm rate (FAR) [%%]\n"
02582         "# $10 = critical success index (CSI) [%%]\n\n");
02583 }
02584
02585 /* Set time interval... */
02586 t0 = t - 0.5 * ctl->dt_mod;
02587 t1 = t + 0.5 * ctl->dt_mod;
02588
02589 /* Initialize grid cells... */
02590 #pragma omp parallel for default(shared) private(ix,iy,iz)
02591 for (ix = 0; ix < ctl->csi_nx; ix++)
02592     for (iy = 0; iy < ctl->csi_ny; iy++)
02593         for (iz = 0; iz < ctl->csi_nz; iz++)
02594             modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
02595
02596 /* Read observation data... */
02597 while (fgets(line, LEN, in)) {
02598
02599     /* Read data... */
02600     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &robs) !=
02601         5)
02602         continue;
02603
02604     /* Check time... */
02605     if (rt < t0)
02606         continue;
02607     if (rt > t1)
02608         break;
02609
02610     /* Calculate indices... */
02611     ix = (int) ((rln - ctl->csi_lon0)
02612         / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02613     iy = (int) ((rln - ctl->csi_lat0)
02614         / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02615     iz = (int) ((rz - ctl->csi_z0)
02616         / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02617
02618     /* Check indices... */
02619     if (ix < 0 || ix >= ctl->csi_nx ||
02620         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02621         continue;
02622
02623     /* Get mean observation index... */
02624     obsmean[ix][iy][iz] += robs;
02625     obscount[ix][iy][iz]++;
02626 }
02627
02628 /* Analyze model data... */
02629 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
02630 for (ip = 0; ip < atm->np; ip++) {
02631
02632     /* Check time... */
02633     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02634         continue;
02635
02636     /* Get indices... */
02637     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
02638         / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02639     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
02640         / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02641     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
02642         / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02643
02644     /* Check indices... */

```

```

02645     if (ix < 0 || ix >= ctl->csi_nx ||
02646         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02647         continue;
02648
02649     /* Get total mass in grid cell... */
02650     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02651 }
02652
02653 /* Analyze all grid cells... */
02654 #pragma omp parallel for default(shared) private(ix,iy,iz,dlon,dlat,lat,area)
02655 for (ix = 0; ix < ctl->csi_nx; ix++)
02656     for (iy = 0; iy < ctl->csi_ny; iy++)
02657         for (iz = 0; iz < ctl->csi_nz; iz++) {
02658
02659             /* Calculate mean observation index... */
02660             if (obscount[ix][iy][iz] > 0)
02661                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
02662
02663             /* Calculate column density... */
02664             if (modmean[ix][iy][iz] > 0) {
02665                 dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
02666                 dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
02667                 lat = ctl->csi_lat0 + dlat * (iy + 0.5);
02668                 area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
02669                     * cos(lat * M_PI / 180.);
02670                 modmean[ix][iy][iz] /= (1e6 * area);
02671             }
02672
02673             /* Calculate CSI... */
02674             if (obscount[ix][iy][iz] > 0) {
02675                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02676                     modmean[ix][iy][iz] >= ctl->csi_modmin)
02677                     cx++;
02678                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02679                     modmean[ix][iy][iz] < ctl->csi_modmin)
02680                     cy++;
02681                 else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
02682                     modmean[ix][iy][iz] >= ctl->csi_modmin)
02683                     cz++;
02684             }
02685         }
02686
02687     /* Write output... */
02688     if (fmod(t, ctl->csi_dt_out) == 0) {
02689
02690         /* Write... */
02691         fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
02692             t, cx, cy, cz, cx + cy, cx + cz,
02693             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
02694             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
02695             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
02696             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
02697
02698         /* Set counters to zero... */
02699         cx = cy = cz = 0;
02700     }
02701
02702     /* Close file... */
02703     if (t == ctl->t_stop)
02704         fclose(out);
02705 }
02706
02707 /*****
02708
02709 void write_ens(
02710     const char *filename,
02711     ctl_t * ctl,
02712     atm_t * atm,
02713     double t) {
02714
02715     static FILE *out;
02716
02717     static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
02718         t0, t1, x[NENS][3], xm[3];
02719
02720     static int ip, iq;
02721
02722     static size_t i, n;
02723
02724     /* Init... */
02725     if (t == ctl->t_start) {
02726
02727         /* Check quantities... */
02728         if (ctl->qnt_ens < 0)
02729             ERRMSG("Missing ensemble IDs!");
02730
02731         /* Create new file... */

```



```

02732     printf("Write ensemble data: %s\n", filename);
02733     if (!(out = fopen(filename, "w")))
02734         ERRMSG("Cannot create file!");
02735
02736     /* Write header... */
02737     fprintf(out,
02738             "# $1 = time [s]\n"
02739             "# $2 = altitude [km]\n"
02740             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02741     for (iq = 0; iq < ctl->nq; iq++)
02742         fprintf(out, "# $d = %s (mean) [%s]\n", 5 + iq,
02743                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02744     for (iq = 0; iq < ctl->nq; iq++)
02745         fprintf(out, "# $d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
02746                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02747     fprintf(out, "# $d = number of members\n\n", 5 + 2 * ctl->nq);
02748 }
02749
02750 /* Set time interval... */
02751 t0 = t - 0.5 * ctl->dt_mod;
02752 t1 = t + 0.5 * ctl->dt_mod;
02753
02754 /* Init... */
02755 ens = GSL_NAN;
02756 n = 0;
02757
02758 /* Loop over air parcels... */
02759 for (ip = 0; ip < atm->np; ip++) {
02760
02761     /* Check time... */
02762     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02763         continue;
02764
02765     /* Check ensemble id... */
02766     if (atm->q[ctl->qnt_ens][ip] != ens) {
02767
02768         /* Write results... */
02769         if (n > 0) {
02770
02771             /* Get mean position... */
02772             xm[0] = xm[1] = xm[2] = 0;
02773             for (i = 0; i < n; i++) {
02774                 xm[0] += x[i][0] / (double) n;
02775                 xm[1] += x[i][1] / (double) n;
02776                 xm[2] += x[i][2] / (double) n;
02777             }
02778             cart2geo(xm, &dummy, &lon, &lat);
02779             fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
02780                     lat);
02781
02782             /* Get quantity statistics... */
02783             for (iq = 0; iq < ctl->nq; iq++) {
02784                 fprintf(out, " ");
02785                 fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02786             }
02787             for (iq = 0; iq < ctl->nq; iq++) {
02788                 fprintf(out, " ");
02789                 fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02790             }
02791             fprintf(out, " %lu\n", n);
02792         }
02793
02794         /* Init new ensemble... */
02795         ens = atm->q[ctl->qnt_ens][ip];
02796         n = 0;
02797     }
02798
02799     /* Save data... */
02800     p[n] = atm->p[ip];
02801     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
02802     for (iq = 0; iq < ctl->nq; iq++)
02803         q[iq][n] = atm->q[iq][ip];
02804     if ((++n) >= NENS)
02805         ERRMSG("Too many data points!");
02806 }
02807
02808 /* Write results... */
02809 if (n > 0) {
02810
02811     /* Get mean position... */
02812     xm[0] = xm[1] = xm[2] = 0;
02813     for (i = 0; i < n; i++) {
02814         xm[0] += x[i][0] / (double) n;
02815         xm[1] += x[i][1] / (double) n;
02816         xm[2] += x[i][2] / (double) n;
02817     }
02818     cart2geo(xm, &dummy, &lon, &lat);

```

```

02819     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
02820
02821     /* Get quantity statistics... */
02822     for (iq = 0; iq < ctl->nq; iq++) {
02823         fprintf(out, " ");
02824         fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02825     }
02826     for (iq = 0; iq < ctl->nq; iq++) {
02827         fprintf(out, " ");
02828         fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02829     }
02830     fprintf(out, " %lu\n", n);
02831 }
02832
02833 /* Close file... */
02834 if (t == ctl->t_stop)
02835     fclose(out);
02836 }
02837
02838 /*****
02839
02840 void write_grid(
02841     const char *filename,
02842     ctl_t * ctl,
02843     met_t * met0,
02844     met_t * met1,
02845     atm_t * atm,
02846     double t) {
02847
02848     FILE *in, *out;
02849
02850     char line[LEN];
02851
02852     static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
02853         area, rho_air, press, temp, cd, vmr, t0, t1, r;
02854
02855     static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec;
02856
02857     /* Check dimensions... */
02858     if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
02859         ERRMSG("Grid dimensions too large!");
02860
02861     /* Set time interval for output... */
02862     t0 = t - 0.5 * ctl->dt_mod;
02863     t1 = t + 0.5 * ctl->dt_mod;
02864
02865     /* Set grid box size... */
02866     dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
02867     dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
02868     dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
02869
02870     /* Initialize grid... */
02871     #pragma omp parallel for default(shared) private(ix,iy,iz)
02872     for (ix = 0; ix < ctl->grid_nx; ix++)
02873         for (iy = 0; iy < ctl->grid_ny; iy++)
02874             for (iz = 0; iz < ctl->grid_nz; iz++) {
02875                 mass[ix][iy][iz] = 0;
02876                 np[ix][iy][iz] = 0;
02877             }
02878
02879     /* Average data... */
02880     #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
02881     for (ip = 0; ip < atm->np; ip++)
02882         if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
02883
02884             /* Get index... */
02885             ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
02886             iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
02887             iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
02888
02889             /* Check indices... */
02890             if (ix < 0 || ix >= ctl->grid_nx ||
02891                 iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
02892                 continue;
02893
02894             /* Add mass... */
02895             if (ctl->qnt_m >= 0)
02896                 mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02897             np[ix][iy][iz]++;
02898         }
02899
02900     /* Check if gnuplot output is requested... */
02901     if (ctl->grid_gpfile[0] != '-') {
02902
02903         /* Write info... */
02904         printf("Plot grid data: %s.png\n", filename);
02905

```

```

02906      /* Create gnuplot pipe... */
02907      if (!(out = popen("gnuplot", "w")))
02908          ERRMSG("Cannot create pipe to gnuplot!");
02909
02910      /* Set plot filename... */
02911      fprintf(out, "set out \"%.s.png\"\\n", filename);
02912
02913      /* Set time string... */
02914      jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02915      fprintf(out, "timestr=\\\"%d-%02d-%02d, %02d:%02d UTC\"\\n",
02916              year, mon, day, hour, min);
02917
02918      /* Dump gnuplot file to pipe... */
02919      if (!(in = fopen(ctl->grid_gpfile, "r")))
02920          ERRMSG("Cannot open file!");
02921      while (fgets(line, LEN, in))
02922          fprintf(out, "%s", line);
02923      fclose(in);
02924  }
02925
02926  else {
02927
02928      /* Write info... */
02929      printf("Write grid data: %s\\n", filename);
02930
02931      /* Create file... */
02932      if (!(out = fopen(filename, "w")))
02933          ERRMSG("Cannot create file!");
02934  }
02935
02936      /* Write header... */
02937      fprintf(out,
02938              "# $1 = time [s]\\n"
02939              "# $2 = altitude [km]\\n"
02940              "# $3 = longitude [deg]\\n"
02941              "# $4 = latitude [deg]\\n"
02942              "# $5 = surface area [km^2]\\n"
02943              "# $6 = layer width [km]\\n"
02944              "# $7 = number of particles [l]\\n"
02945              "# $8 = column density [kg/m^2]\\n"
02946              "# $9 = volume mixing ratio [l]\\n\\n");
02947
02948      /* Write data... */
02949      for (ix = 0; ix < ctl->grid_nx; ix++) {
02950          if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
02951              fprintf(out, "\\n");
02952          for (iy = 0; iy < ctl->grid_ny; iy++) {
02953              if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
02954                  fprintf(out, "\\n");
02955              for (iz = 0; iz < ctl->grid_nz; iz++)
02956                  if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
02957
02958                      /* Set coordinates... */
02959                      z = ctl->grid_z0 + dz * (iz + 0.5);
02960                      lon = ctl->grid_lon0 + dlon * (ix + 0.5);
02961                      lat = ctl->grid_lat0 + dlat * (iy + 0.5);
02962
02963                      /* Get pressure and temperature... */
02964                      press = P(z);
02965                      intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
02966                                      NULL, &temp, NULL, NULL, NULL, NULL, NULL, NULL);
02967
02968                      /* Calculate surface area... */
02969                      area = dlat * dlon * SQR(RE * M_PI / 180.)
02970                          * cos(lat * M_PI / 180.);
02971
02972                      /* Calculate column density... */
02973                      cd = mass[ix][iy][iz] / (1e6 * area);
02974
02975                      /* Calculate volume mixing ratio... */
02976                      rho_air = 100. * press / (RA * temp);
02977                      vmr = MA / ctl->molmass * mass[ix][iy][iz]
02978                          / (rho_air * 1e6 * area * 1e3 * dz);
02979
02980                      /* Write output... */
02981                      fprintf(out, "%.2f %g %g %g %g %d %g %g\\n",
02982                              t, z, lon, lat, area, dz, np[ix][iy][iz], cd, vmr);
02983                  }
02984              }
02985          }
02986
02987      /* Close file... */
02988      fclose(out);
02989  }
02990
02991  /*****
02992

```

```

02993 void write_prof(
02994     const char *filename,
02995     ctl_t *ctl,
02996     met_t *met0,
02997     met_t *met1,
02998     atm_t *atm,
02999     double t) {
03000
03001     static FILE *in, *out;
03002
03003     static char line[LEN];
03004
03005     static double mass[GX][GY][GZ], obsmean[GX][GY], obsmean2[GX][GY], rt, rz,
03006         rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp,
03007         rho_air, vmr, h2o, o3;
03008
03009     static int obscount[GX][GY], ip, ix, iy, iz, okay;
03010
03011     /* Init... */
03012     if (t == ctl->t_start) {
03013
03014         /* Check quantity index for mass... */
03015         if (ctl->qnt_m < 0)
03016             ERRMSG("Need quantity mass!");
03017
03018         /* Check dimensions... */
03019         if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
03020             ERRMSG("Grid dimensions too large!");
03021
03022         /* Open observation data file... */
03023         printf("Read profile observation data: %s\n", ctl->prof_obsfile);
03024         if (!(in = fopen(ctl->prof_obsfile, "r")))
03025             ERRMSG("Cannot open file!");
03026
03027         /* Create new output file... */
03028         printf("Write profile data: %s\n", filename);
03029         if (!(out = fopen(filename, "w")))
03030             ERRMSG("Cannot create file!");
03031
03032         /* Write header... */
03033         fprintf(out,
03034             "# $1 = time [s]\n"
03035             "# $2 = altitude [km]\n"
03036             "# $3 = longitude [deg]\n"
03037             "# $4 = latitude [deg]\n"
03038             "# $5 = pressure [hPa]\n"
03039             "# $6 = temperature [K]\n"
03040             "# $7 = volume mixing ratio [1]\n"
03041             "# $8 = H2O volume mixing ratio [1]\n"
03042             "# $9 = O3 volume mixing ratio [1]\n"
03043             "# $10 = observed BT index (mean) [K]\n"
03044             "# $11 = observed BT index (sigma) [K]\n");
03045
03046         /* Set grid box size... */
03047         dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
03048         dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
03049         dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
03050     }
03051
03052     /* Set time interval... */
03053     t0 = t - 0.5 * ctl->dt_mod;
03054     t1 = t + 0.5 * ctl->dt_mod;
03055
03056     /* Initialize... */
03057     #pragma omp parallel for default(shared) private(ix,iy,iz)
03058     for (ix = 0; ix < ctl->prof_nx; ix++)
03059         for (iy = 0; iy < ctl->prof_ny; iy++) {
03060             obsmean[ix][iy] = 0;
03061             obsmean2[ix][iy] = 0;
03062             obscount[ix][iy] = 0;
03063             for (iz = 0; iz < ctl->prof_nz; iz++)
03064                 mass[ix][iy][iz] = 0;
03065         }
03066
03067     /* Read observation data... */
03068     while (fgets(line, LEN, in)) {
03069
03070         /* Read data... */
03071         if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
03072             5)
03073             continue;
03074
03075         /* Check time... */
03076         if (rt < t0)
03077             continue;
03078         if (rt > t1)
03079             break;

```

```

03080
03081     /* Calculate indices... */
03082     ix = (int) ((rlon - ctl->prof_lon0) / dlon);
03083     iy = (int) ((rlat - ctl->prof_lat0) / dlat);
03084
03085     /* Check indices... */
03086     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
03087         continue;
03088
03089     /* Get mean observation index... */
03090     obsmean[ix][iy] += robs;
03091     obsmean2[ix][iy] += SQR(robs);
03092     obscount[ix][iy]++;
03093 }
03094
03095 /* Analyze model data... */
03096 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
03097 for (ip = 0; ip < atm->np; ip++) {
03098
03099     /* Check time... */
03100     if (atm->time[ip] < t0 || atm->time[ip] > t1)
03101         continue;
03102
03103     /* Get indices... */
03104     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
03105     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
03106     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
03107
03108     /* Check indices... */
03109     if (ix < 0 || ix >= ctl->prof_nx ||
03110         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
03111         continue;
03112
03113     /* Get total mass in grid cell... */
03114     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
03115 }
03116
03117 /* Extract profiles... */
03118 for (ix = 0; ix < ctl->prof_nx; ix++)
03119     for (iy = 0; iy < ctl->prof_ny; iy++)
03120         if (obscount[ix][iy] > 0) {
03121
03122             /* Check profile... */
03123             okay = 0;
03124             for (iz = 0; iz < ctl->prof_nz; iz++)
03125                 if (mass[ix][iy][iz] > 0) {
03126                     okay = 1;
03127                     break;
03128                 }
03129             if (!okay)
03130                 continue;
03131
03132             /* Write output... */
03133             fprintf(out, "\n");
03134
03135             /* Loop over altitudes... */
03136             for (iz = 0; iz < ctl->prof_nz; iz++) {
03137
03138                 /* Set coordinates... */
03139                 z = ctl->prof_z0 + dz * (iz + 0.5);
03140                 lon = ctl->prof_lon0 + dlon * (ix + 0.5);
03141                 lat = ctl->prof_lat0 + dlat * (iy + 0.5);
03142
03143                 /* Get pressure and temperature... */
03144                 press = P(z);
03145                 intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
03146                               NULL, &temp, NULL, NULL, NULL, NULL, &h2o, &o3);
03147
03148                 /* Calculate surface area... */
03149                 area = dlat * dlon * SQR(M_PI * RE / 180.)
03150                     * cos(lat * M_PI / 180.);
03151
03152                 /* Calculate volume mixing ratio... */
03153                 rho_air = 100. * press / (RA * temp);
03154                 vmr = MA / ctl->molmass * mass[ix][iy][iz]
03155                     / (rho_air * area * dz * 1e9);
03156
03157                 /* Write output... */
03158                 fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
03159                     t, z, lon, lat, press, temp, vmr, h2o, o3,
03160                     obsmean[ix][iy] / obscount[ix][iy],
03161                     sqrt(obsmean2[ix][iy] / obscount[ix][iy]
03162                         - SQR(obsmean[ix][iy] / obscount[ix][iy])));
03163             }
03164         }
03165
03166     /* Close file... */

```

```

03167     if (t == ctl->t_stop)
03168         fclose(out);
03169 }
03170
03171 /*****
03172
03173 void write_station(
03174     const char *filename,
03175     ctl_t * ctl,
03176     atm_t * atm,
03177     double t) {
03178
03179     static FILE *out;
03180
03181     static double rmax2, t0, t1, x0[3], x1[3];
03182
03183     /* Init... */
03184     if (t == ctl->t_start) {
03185
03186         /* Write info... */
03187         printf("Write station data: %s\n", filename);
03188
03189         /* Create new file... */
03190         if (!(out = fopen(filename, "w")))
03191             ERRMSG("Cannot create file!");
03192
03193         /* Write header... */
03194         fprintf(out,
03195             "# $1 = time [s]\n"
03196             "# $2 = altitude [km]\n"
03197             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03198         for (int iq = 0; iq < ctl->nq; iq++)
03199             fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
03200                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
03201         fprintf(out, "\n");
03202
03203         /* Set geolocation and search radius... */
03204         geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
03205         rmax2 = SQR(ctl->stat_r);
03206     }
03207
03208     /* Set time interval for output... */
03209     t0 = t - 0.5 * ctl->dt_mod;
03210     t1 = t + 0.5 * ctl->dt_mod;
03211
03212     /* Loop over air parcels... */
03213     for (int ip = 0; ip < atm->np; ip++) {
03214
03215         /* Check time... */
03216         if (atm->time[ip] < t0 || atm->time[ip] > t1)
03217             continue;
03218
03219         /* Check station flag... */
03220         if (ctl->qnt_stat >= 0)
03221             if (atm->q[ctl->qnt_stat][ip])
03222                 continue;
03223
03224         /* Get Cartesian coordinates... */
03225         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
03226
03227         /* Check horizontal distance... */
03228         if (DIST2(x0, x1) > rmax2)
03229             continue;
03230
03231         /* Set station flag... */
03232         if (ctl->qnt_stat >= 0)
03233             atm->q[ctl->qnt_stat][ip] = 1;
03234
03235         /* Write data... */
03236         fprintf(out, "%.2f %g %g %g",
03237             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
03238         for (int iq = 0; iq < ctl->nq; iq++) {
03239             fprintf(out, " ");
03240             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
03241         }
03242         fprintf(out, "\n");
03243     }
03244
03245     /* Close file... */
03246     if (t == ctl->t_stop)
03247         fclose(out);
03248 }

```

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.
- double [clim_hno3](#) (double t, double lat, double p)
Climatology of HNO₃ volume mixing ratios.
- 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.
- double [intpol_met_2d](#) (double array[EX][EY], int ix, int iy, double wx, double wy)
Linear interpolation of 2-D meteorological data.
- double [intpol_met_3d](#) (float array[EX][EY][EP], int ip, int ix, int iy, double wp, double wx, double wy)
Linear interpolation of 3-D meteorological data.
- void [intpol_met_space](#) ([met_t](#) *met, double p, double lon, double lat, double *ps, double *pt, double *z, double *t, double *u, double *v, double *w, double *pv, double *h2o, double *o3)
Spatial interpolation of meteorological data.
- void [intpol_met_time](#) ([met_t](#) *met0, [met_t](#) *met1, double ts, double p, double lon, double lat, double *ps, double *pt, double *z, double *t, double *u, double *v, double *w, double *pv, double *h2o, double *o3)
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_extrapolate](#) ([met_t](#) *met)
Extrapolate meteorological data at lower boundary.
- void [read_met_geopot](#) ([ctl_t](#) *ctl, [met_t](#) *met)
Calculate geopotential heights.
- void [read_met_help](#) (int ncid, char *varname, char *varname2, [met_t](#) *met, float dest[EX][EY][EP], float scl)
Read and convert 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_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 }
```

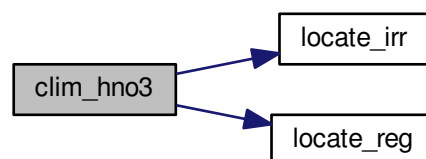
5.21.2.2 double clim_hno3 (double t, double lat, double p)

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
00303     /* Get indices... */
00304     int isec = locate_irr(clim_hno3_secs, 12, sec);
00305     int ilat = locate_reg(clim_hno3_lats, 18, lat);
00306     int ip = locate_irr(clim_hno3_ps, 10, p);
00307
00308     /* Interpolate... */
00309     double aux00 = LIN(clim_hno3_ps[ip], clim_hno3_var[isec][ilat][ip],
00310                       clim_hno3_ps[ip + 1], clim_hno3_var[isec][ilat][ip + 1],
00311                       p);
00312     double aux01 = LIN(clim_hno3_ps[ip], clim_hno3_var[isec][ilat + 1][ip],
00313                       clim_hno3_ps[ip + 1], clim_hno3_var[isec][ilat + 1][ip],
00314                       p);
00315     double aux10 = LIN(clim_hno3_ps[ip], clim_hno3_var[isec + 1][ilat][ip],
00316                       clim_hno3_ps[ip + 1], clim_hno3_var[isec + 1][ilat][ip],
00317                       p);
00318     double aux11 = LIN(clim_hno3_ps[ip], clim_hno3_var[isec + 1][ilat + 1][ip],
00319                       clim_hno3_ps[ip + 1], clim_hno3_var[isec + 1][ilat + 1][ip],
00320                       p);
00321     aux00 =
00322     LIN(clim_hno3_lats[ilat], aux00, clim_hno3_lats[ilat + 1], aux01, lat);
00323     aux11 =
00324     LIN(clim_hno3_lats[ilat], aux10, clim_hno3_lats[ilat + 1], aux11, lat);
00325     return LIN(clim_hno3_secs[isec], aux00, clim_hno3_secs[isec + 1], aux11,
00326               sec);
00327 }
00328 }
```

Here is the call graph for this function:



5.21.2.3 double clim_tropo (double *t*, double *lat*)

Climatology of tropopause pressure.

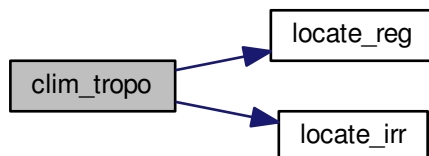
Definition at line 457 of file [libtrac.c](#).

```

00459         {
00460
00461     /* Get day of year... */
00462     double doy = FMOD(t / 86400., 365.25);
00463     while (doy < 0)
00464         doy += 365.25;
00465
00466     /* Get indices... */
00467     int ilat = locate_reg(clim_tropo_lats, 73, lat);
00468     int imon = locate_irr(clim_tropo_doys, 12, doy);
00469
00470     /* Interpolate... */
00471     double p0 = LIN(clim_tropo_lats[ilat], clim_tropo_tps[imon][ilat],
00472                    clim_tropo_lats[ilat + 1], clim_tropo_tps[imon][ilat + 1],
00473                    lat);
00474     double p1 = LIN(clim_tropo_lats[ilat], clim_tropo_tps[imon + 1][ilat],
00475                    clim_tropo_lats[ilat + 1],
00476                    clim_tropo_tps[imon + 1][ilat + 1],
00477                    lat);
00478     return LIN(clim_tropo_doys[imon], p0, clim_tropo_doys[imon + 1], p1, doy);
00479 }

```

Here is the call graph for this function:



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

Get day of year from date.

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

```

00487         {
00488
00489     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00490     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00491
00492     /* Get day of year... */
00493     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
00494         *doy = d0l[mon - 1] + day - 1;
00495     else
00496         *doy = d0[mon - 1] + day - 1;
00497 }

```

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

Get date from day of year.

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

```
00505         {
00506
00507     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00508     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00509     int i;
00510
00511     /* Get month and day... */
00512     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
00513         for (i = 11; i >= 0; i--)
00514             if (d0l[i] <= doy)
00515                 break;
00516         *mon = i + 1;
00517         *day = doy - d0l[i] + 1;
00518     } else {
00519         for (i = 11; i >= 0; i--)
00520             if (d0[i] <= doy)
00521                 break;
00522         *mon = i + 1;
00523         *day = doy - d0[i] + 1;
00524     }
00525 }
```

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

Convert geolocation to Cartesian coordinates.

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

```
00533         {
00534
00535     double radius = z + RE;
00536     x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
00537     x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
00538     x[2] = radius * sin(lat / 180 * M_PI);
00539 }
```

5.21.2.7 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 543 of file [libtrac.c](#).

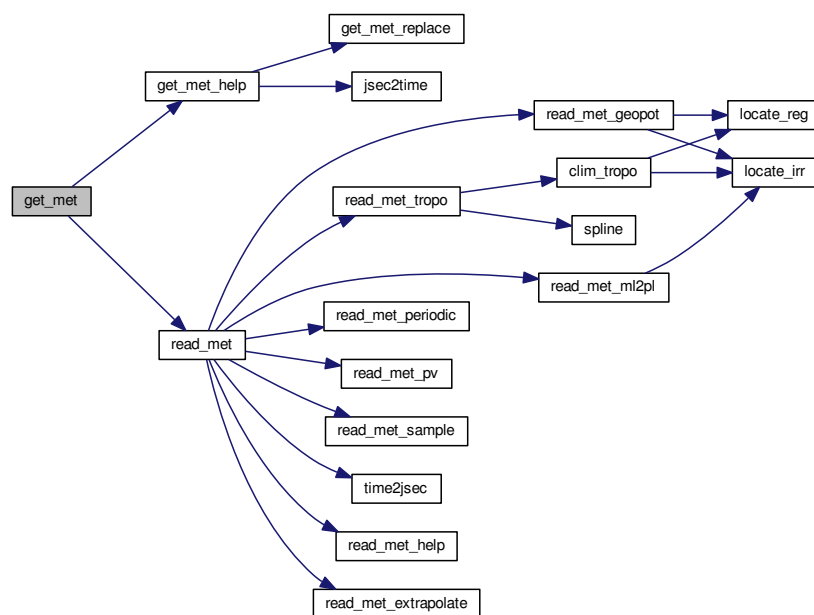
```
00548         {
00549
00550     static int init, ip, ix, iy;
00551     met_t *mets;
00552
00553     char filename[LEN];
00554
00555     /* Init... */
00556     if (t == ctl->t_start || !init) {
00557         init = 1;
00558
00559         get_met_help(t, -1, metbase, ctl->dt_met, filename);
00560         if (!read_met(ctl, filename, *met0))
00561             ERRMSG("Cannot open file!");
00562
00563         get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
00564 dt_met, filename);
00565         if (!read_met(ctl, filename, *met1))
00566             ERRMSG("Cannot open file!");
00567 #ifdef _OPENACC
00568         met_t *met0up = *met0;
```

```

00569     met_t *metlup = *met1;
00570 #pragma acc update device(met0up[:1],metlup[:1])
00571 #endif
00572 }
00573
00574 /* Read new data for forward trajectories... */
00575 if (t > (*met1)->time && ctl->direction == 1) {
00576     mets = *met1;
00577     *met1 = *met0;
00578     *met0 = mets;
00579     get_met_help(t, 1, metbase, ctl->dt_met, filename);
00580     if (!read_met(ctl, filename, *met1))
00581         ERRMSG("Cannot open file!");
00582 #ifdef _OPENACC
00583     met_t *metlup = *met1;
00584 #pragma acc update device(metlup[:1])
00585 #endif
00586 }
00587
00588 /* Read new data for backward trajectories... */
00589 if (t < (*met0)->time && ctl->direction == -1) {
00590     mets = *met1;
00591     *met1 = *met0;
00592     *met0 = mets;
00593     get_met_help(t, -1, metbase, ctl->dt_met, filename);
00594     if (!read_met(ctl, filename, *met0))
00595         ERRMSG("Cannot open file!");
00596 #ifdef _OPENACC
00597     met_t *met0up = *met0;
00598 #pragma acc update device(met0up[:1])
00599 #endif
00600 }
00601
00602 /* Check that grids are consistent... */
00603 if ((*met0)->nx != (*met1)->nx
00604     || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
00605     ERRMSG("Meteo grid dimensions do not match!");
00606 for (ix = 0; ix < (*met0)->nx; ix++)
00607     if ((*met0)->lon[ix] != (*met1)->lon[ix])
00608         ERRMSG("Meteo grid longitudes do not match!");
00609 for (iy = 0; iy < (*met0)->ny; iy++)
00610     if ((*met0)->lat[iy] != (*met1)->lat[iy])
00611         ERRMSG("Meteo grid latitudes do not match!");
00612 for (ip = 0; ip < (*met0)->np; ip++)
00613     if ((*met0)->p[ip] != (*met1)->p[ip])
00614         ERRMSG("Meteo grid pressure levels do not match!");
00615 }

```

Here is the call graph for this function:



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

Get meteorological data for timestep.

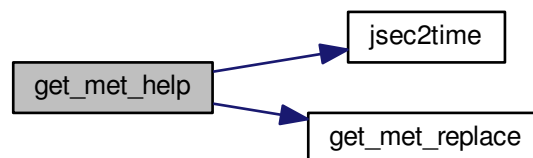
Definition at line 619 of file [libtrac.c](#).

```

00624         {
00625
00626     char repl[LEN];
00627
00628     double t6, r;
00629
00630     int year, mon, day, hour, min, sec;
00631
00632     /* Round time to fixed intervals... */
00633     if (direct == -1)
00634         t6 = floor(t / dt_met) * dt_met;
00635     else
00636         t6 = ceil(t / dt_met) * dt_met;
00637
00638     /* Decode time... */
00639     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00640
00641     /* Set filename... */
00642     sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
00643     sprintf(repl, "%d", year);
00644     get_met_replace(filename, "YYYY", repl);
00645     sprintf(repl, "%02d", mon);
00646     get_met_replace(filename, "MM", repl);
00647     sprintf(repl, "%02d", day);
00648     get_met_replace(filename, "DD", repl);
00649     sprintf(repl, "%02d", hour);
00650     get_met_replace(filename, "HH", repl);
00651 }

```

Here is the call graph for this function:



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

Replace template strings in filename.

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

```

00658         {
00659
00660     char buffer[LEN], *ch;
00661
00662     /* Iterate... */
00663     for (int i = 0; i < 3; i++) {
00664
00665         /* Replace substring... */
00666         if (!(ch = strstr(orig, search)))
00667             return;
00668         strncpy(buffer, orig, (size_t) (ch - orig));
00669         buffer[ch - orig] = 0;
00670         sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
00671         orig[0] = 0;
00672         strcpy(orig, buffer);
00673     }
00674 }

```

5.21.2.10 double intpol_met_2d (double array[EX][EY], int ix, int iy, double wx, double wy)

Linear interpolation of 2-D meteorological data.

Definition at line 678 of file [libtrac.c](#).

```
00683     {
00684
00685     /* Set variables... */
00686     double aux00 = array[ix][iy];
00687     double aux01 = array[ix][iy + 1];
00688     double aux10 = array[ix + 1][iy];
00689     double aux11 = array[ix + 1][iy + 1];
00690
00691     /* Interpolate horizontally... */
00692     aux00 = wy * (aux00 - aux01) + aux01;
00693     aux11 = wy * (aux10 - aux11) + aux11;
00694     return wx * (aux00 - aux11) + aux11;
00695 }
```

5.21.2.11 double intpol_met_3d (float array[EX][EY][EP], int ip, int ix, int iy, double wp, double wx, double wy)

Linear interpolation of 3-D meteorological data.

Definition at line 699 of file [libtrac.c](#).

```
00706     {
00707
00708     /* Interpolate vertically... */
00709     double aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00710     + array[ix][iy][ip + 1];
00711     double aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
00712     + array[ix][iy + 1][ip + 1];
00713     double aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
00714     + array[ix + 1][iy][ip + 1];
00715     double aux11 =
00716     wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
00717     + array[ix + 1][iy + 1][ip + 1];
00718
00719     /* Interpolate horizontally... */
00720     aux00 = wy * (aux00 - aux01) + aux01;
00721     aux11 = wy * (aux10 - aux11) + aux11;
00722     return wx * (aux00 - aux11) + aux11;
00723 }
```

5.21.2.12 void intpol_met_space (met_t * met, double p, double lon, double lat, double * ps, double * pt, double * z, double * t, double * u, double * v, double * w, double * pv, double * h2o, double * o3)

Spatial interpolation of meteorological data.

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

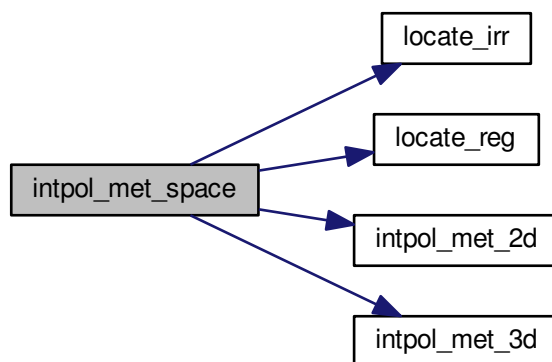
```
00741     {
00742
00743     /* Check longitude... */
00744     if (met->lon[met->nx - 1] > 180 && lon < 0)
00745         lon += 360;
00746
00747     /* Get indices... */
00748     int ip = locate_irr(met->p, met->np, p);
00749     int ix = locate_reg(met->lon, met->nx, lon);
00750     int iy = locate_reg(met->lat, met->ny, lat);
00751
00752     /* Get weights... */
00753     double wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
00754     double wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
00755     double wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);
00756
00757     /* Interpolate... */
```

```

00758     if (ps != NULL)
00759         *ps = intpol_met_2d(met->ps, ix, iy, wx, wy);
00760     if (pt != NULL)
00761         *pt = intpol_met_2d(met->pt, ix, iy, wx, wy);
00762     if (z != NULL)
00763         *z = intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy);
00764     if (t != NULL)
00765         *t = intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy);
00766     if (u != NULL)
00767         *u = intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy);
00768     if (v != NULL)
00769         *v = intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy);
00770     if (w != NULL)
00771         *w = intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy);
00772     if (pv != NULL)
00773         *pv = intpol_met_3d(met->pv, ip, ix, iy, wp, wx, wy);
00774     if (h2o != NULL)
00775         *h2o = intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy);
00776     if (o3 != NULL)
00777         *o3 = intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy);
00778 }

```

Here is the call graph for this function:



5.21.2.13 `void intpol_met_time (met_t * met0, met_t * met1, double ts, double p, double lon, double lat, double * ps, double * pt, double * z, double * t, double * u, double * v, double * w, double * pv, double * h2o, double * o3)`

Temporal interpolation of meteorological data.

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

```

00798     {
00799
00800         double h2o0, h2o1, o30, o31, ps0, ps1, pt0, pt1, pv0, pv1, t0, t1, u0, u1,
00801             v0, v1, w0, w1, wt, z0, z1;
00802
00803         /* Spatial interpolation... */
00804         intpol_met_space(met0, p, lon, lat,
00805             ps == NULL ? NULL : &ps0,
00806             pt == NULL ? NULL : &pt0,
00807             z == NULL ? NULL : &z0,
00808             t == NULL ? NULL : &t0,
00809             u == NULL ? NULL : &u0,
00810             v == NULL ? NULL : &v0,
00811             w == NULL ? NULL : &w0,
00812             pv == NULL ? NULL : &pv0,
00813             h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00814         intpol_met_space(met1, p, lon, lat,

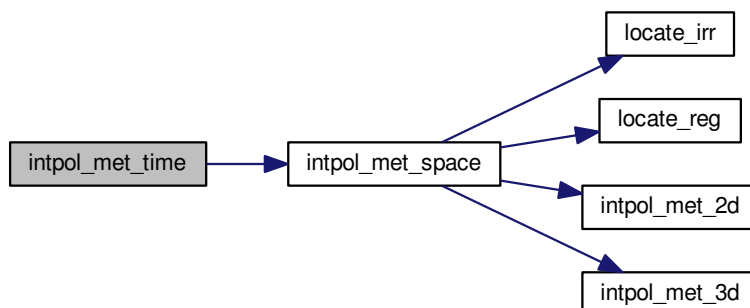
```

```

00815         ps == NULL ? NULL : &ps1,
00816         pt == NULL ? NULL : &pt1,
00817         z == NULL ? NULL : &z1,
00818         t == NULL ? NULL : &t1,
00819         u == NULL ? NULL : &u1,
00820         v == NULL ? NULL : &v1,
00821         w == NULL ? NULL : &w1,
00822         pv == NULL ? NULL : &pv1,
00823         h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00824
00825     /* Get weighting factor... */
00826     wt = (met1->time - ts) / (met1->time - met0->time);
00827
00828     /* Interpolate... */
00829     if (ps != NULL)
00830         *ps = wt * (ps0 - ps1) + ps1;
00831     if (pt != NULL)
00832         *pt = wt * (pt0 - pt1) + pt1;
00833     if (z != NULL)
00834         *z = wt * (z0 - z1) + z1;
00835     if (t != NULL)
00836         *t = wt * (t0 - t1) + t1;
00837     if (u != NULL)
00838         *u = wt * (u0 - u1) + u1;
00839     if (v != NULL)
00840         *v = wt * (v0 - v1) + v1;
00841     if (w != NULL)
00842         *w = wt * (w0 - w1) + w1;
00843     if (pv != NULL)
00844         *pv = wt * (pv0 - pv1) + pv1;
00845     if (h2o != NULL)
00846         *h2o = wt * (h2o0 - h2o1) + h2o1;
00847     if (o3 != NULL)
00848         *o3 = wt * (o30 - o31) + o31;
00849 }

```

Here is the call graph for this function:



5.21.2.14 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 853 of file `libtrac.c`.

```

00861     {
00862
00863     struct tm t0, *t1;
00864
00865     t0.tm_year = 100;
00866     t0.tm_mon = 0;
00867     t0.tm_mday = 1;
00868     t0.tm_hour = 0;

```



```

00869     t0.tm_min = 0;
00870     t0.tm_sec = 0;
00871
00872     time_t jsec0 = (time_t) jsec + timegm(&t0);
00873     t1 = gmtime(&jsec0);
00874
00875     *year = t1->tm_year + 1900;
00876     *mon = t1->tm_mon + 1;
00877     *day = t1->tm_mday;
00878     *hour = t1->tm_hour;
00879     *min = t1->tm_min;
00880     *sec = t1->tm_sec;
00881     *remain = jsec - floor(jsec);
00882 }

```

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

Find array index for irregular grid.

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

```

00889     {
00890
00891     int ilo = 0;
00892     int ihi = n - 1;
00893     int i = (ihi + ilo) >> 1;
00894
00895     if (xx[i] < xx[i + 1])
00896         while (ihi > ilo + 1) {
00897             i = (ihi + ilo) >> 1;
00898             if (xx[i] > x)
00899                 ihi = i;
00900             else
00901                 ilo = i;
00902         } else
00903         while (ihi > ilo + 1) {
00904             i = (ihi + ilo) >> 1;
00905             if (xx[i] <= x)
00906                 ihi = i;
00907             else
00908                 ilo = i;
00909         }
00910
00911     return ilo;
00912 }

```

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

Find array index for regular grid.

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

```

00919     {
00920
00921     /* Calculate index... */
00922     int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
00923
00924     /* Check range... */
00925     if (i < 0)
00926         i = 0;
00927     else if (i >= n - 2)
00928         i = n - 2;
00929
00930     return i;
00931 }

```

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

Read atmospheric data.

Definition at line 935 of file [libtrac.c](#).

```

00938         {
00939
00940     FILE *in;
00941
00942     char line[LEN], *tok;
00943
00944     double t0;
00945
00946     int dimid, ip, iq, ncid, varid;
00947
00948     size_t nparts;
00949
00950     /* Init... */
00951     atm->np = 0;
00952
00953     /* Write info... */
00954     printf("Read atmospheric data: %s\n", filename);
00955
00956     /* Read ASCII data... */
00957     if (ctl->atm_type == 0) {
00958
00959         /* Open file... */
00960         if (!(in = fopen(filename, "r"))) {
00961             WARN("File not found!");
00962             return 0;
00963         }
00964
00965         /* Read line... */
00966         while (fgets(line, LEN, in)) {
00967
00968             /* Read data... */
00969             TOK(line, tok, "%lg", atm->time[atm->np]);
00970             TOK(NULL, tok, "%lg", atm->p[atm->np]);
00971             TOK(NULL, tok, "%lg", atm->lon[atm->np]);
00972             TOK(NULL, tok, "%lg", atm->lat[atm->np]);
00973             for (iq = 0; iq < ctl->nq; iq++)
00974                 TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00975
00976             /* Convert altitude to pressure... */
00977             atm->p[atm->np] = P(atm->p[atm->np]);
00978
00979             /* Increment data point counter... */
00980             if (++atm->np > NP)
00981                 ERRMSG("Too many data points!");
00982         }
00983
00984         /* Close file... */
00985         fclose(in);
00986     }
00987
00988     /* Read binary data... */
00989     else if (ctl->atm_type == 1) {
00990
00991         /* Open file... */
00992         if (!(in = fopen(filename, "r")))
00993             return 0;
00994
00995         /* Read data... */
00996         FREAD(&atm->np, int, 1, in);
00997         FREAD(atm->time, double,
00998             (size_t) atm->np,
00999             in);
01000         FREAD(atm->p, double,
01001             (size_t) atm->np,
01002             in);
01003         FREAD(atm->lon, double,
01004             (size_t) atm->np,
01005             in);
01006         FREAD(atm->lat, double,
01007             (size_t) atm->np,
01008             in);
01009         for (iq = 0; iq < ctl->nq; iq++)
01010             FREAD(atm->q[iq], double,
01011                 (size_t) atm->np,
01012                 in);
01013
01014         /* Close file... */

```

```

01015     fclose(in);
01016 }
01017
01018 /* Read netCDF data... */
01019 else if (ctl->atm_type == 2) {
01020
01021     /* Open file... */
01022     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
01023         return 0;
01024
01025     /* Get dimensions... */
01026     NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
01027     NC(nc_inq_dimlen(ncid, dimid, &nparts));
01028     atm->np = (int) nparts;
01029     if (atm->np > NP)
01030         ERRMSG("Too many particles!");
01031
01032     /* Get time... */
01033     NC(nc_inq_varid(ncid, "time", &varid));
01034     NC(nc_get_var_double(ncid, varid, &t0));
01035     for (ip = 0; ip < atm->np; ip++)
01036         atm->time[ip] = t0;
01037
01038     /* Read geolocations... */
01039     NC(nc_inq_varid(ncid, "PRESS", &varid));
01040     NC(nc_get_var_double(ncid, varid, atm->p));
01041     NC(nc_inq_varid(ncid, "LON", &varid));
01042     NC(nc_get_var_double(ncid, varid, atm->lon));
01043     NC(nc_inq_varid(ncid, "LAT", &varid));
01044     NC(nc_get_var_double(ncid, varid, atm->lat));
01045
01046     /* Read variables... */
01047     if (ctl->qnt_p >= 0)
01048         if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
01049             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
01050     if (ctl->qnt_t >= 0)
01051         if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
01052             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
01053     if (ctl->qnt_u >= 0)
01054         if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
01055             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
01056     if (ctl->qnt_v >= 0)
01057         if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
01058             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
01059     if (ctl->qnt_w >= 0)
01060         if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
01061             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
01062     if (ctl->qnt_h2o >= 0)
01063         if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
01064             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
01065     if (ctl->qnt_o3 >= 0)
01066         if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
01067             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
01068     if (ctl->qnt_theta >= 0)
01069         if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
01070             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
01071     if (ctl->qnt_pv >= 0)
01072         if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
01073             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
01074
01075     /* Check data... */
01076     for (ip = 0; ip < atm->np; ip++)
01077         if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
01078             || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
01079             || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
01080             || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
01081             || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
01082         atm->time[ip] = GSL_NAN;
01083         atm->p[ip] = GSL_NAN;
01084         atm->lon[ip] = GSL_NAN;
01085         atm->lat[ip] = GSL_NAN;
01086         for (iq = 0; iq < atm->nq; iq++)
01087             atm->q[iq][ip] = GSL_NAN;
01088     } else {
01089         if (ctl->qnt_h2o >= 0)
01090             atm->q[ctl->qnt_h2o][ip] *= 1.608;
01091         if (ctl->qnt_pv >= 0)
01092             atm->q[ctl->qnt_pv][ip] *= 1e6;
01093         if (atm->lon[ip] > 180)
01094             atm->lon[ip] -= 360;
01095     }
01096
01097     /* Close file... */
01098     NC(nc_close(ncid));
01099 }
01100
01101 /* Error... */

```

```

01102     else
01103         ERRMSG("Atmospheric data type not supported!");
01104
01105     /* Check number of points... */
01106     if (atm->np < 1)
01107         ERRMSG("Can not read any data!");
01108
01109     /* Return success... */
01110     return 1;
01111 }

```

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

Read control parameters.

Definition at line 1115 of file libtrac.c.

```

01119     {
01120
01121         /* Write info... */
01122         printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
01123             "(executable: %s | compiled: %s, %s)\n\n",
01124             argv[0], __DATE__, __TIME__);
01125
01126         /* Initialize quantity indices... */
01127         ctl->qnt_ens = -1;
01128         ctl->qnt_m = -1;
01129         ctl->qnt_rho = -1;
01130         ctl->qnt_ps = -1;
01131         ctl->qnt_pt = -1;
01132         ctl->qnt_z = -1;
01133         ctl->qnt_p = -1;
01134         ctl->qnt_t = -1;
01135         ctl->qnt_u = -1;
01136         ctl->qnt_v = -1;
01137         ctl->qnt_w = -1;
01138         ctl->qnt_h2o = -1;
01139         ctl->qnt_o3 = -1;
01140         ctl->qnt_theta = -1;
01141         ctl->qnt_vh = -1;
01142         ctl->qnt_vz = -1;
01143         ctl->qnt_pv = -1;
01144         ctl->qnt_tice = -1;
01145         ctl->qnt_tsts = -1;
01146         ctl->qnt_tnat = -1;
01147         ctl->qnt_stat = -1;
01148
01149         /* Read quantities... */
01150         ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
01151         if (ctl->nq > NQ)
01152             ERRMSG("Too many quantities!");
01153         for (int iq = 0; iq < ctl->nq; iq++) {
01154
01155             /* Read quantity name and format... */
01156             scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
01157             scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
01158                 ctl->qnt_format[iq]);
01159
01160             /* Try to identify quantity... */
01161             if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
01162                 ctl->qnt_ens = iq;
01163                 sprintf(ctl->qnt_unit[iq], "-");
01164             } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
01165                 ctl->qnt_m = iq;
01166                 sprintf(ctl->qnt_unit[iq], "kg");
01167             } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
01168                 ctl->qnt_rho = iq;
01169                 sprintf(ctl->qnt_unit[iq], "m");
01170             } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
01171                 ctl->qnt_rho = iq;
01172                 sprintf(ctl->qnt_unit[iq], "kg/m^3");
01173             } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
01174                 ctl->qnt_ps = iq;
01175                 sprintf(ctl->qnt_unit[iq], "hPa");
01176             } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
01177                 ctl->qnt_pt = iq;
01178                 sprintf(ctl->qnt_unit[iq], "hPa");
01179             } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
01180                 ctl->qnt_z = iq;
01181             }
01182         }
01183     }

```

```

01182     sprintf(ctl->qnt_unit[iq], "km");
01183 } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
01184     ctl->qnt_p = iq;
01185     sprintf(ctl->qnt_unit[iq], "hPa");
01186 } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
01187     ctl->qnt_t = iq;
01188     sprintf(ctl->qnt_unit[iq], "K");
01189 } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
01190     ctl->qnt_u = iq;
01191     sprintf(ctl->qnt_unit[iq], "m/s");
01192 } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
01193     ctl->qnt_v = iq;
01194     sprintf(ctl->qnt_unit[iq], "m/s");
01195 } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
01196     ctl->qnt_w = iq;
01197     sprintf(ctl->qnt_unit[iq], "hPa/s");
01198 } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
01199     ctl->qnt_h2o = iq;
01200     sprintf(ctl->qnt_unit[iq], "l");
01201 } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
01202     ctl->qnt_o3 = iq;
01203     sprintf(ctl->qnt_unit[iq], "l");
01204 } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
01205     ctl->qnt_theta = iq;
01206     sprintf(ctl->qnt_unit[iq], "K");
01207 } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
01208     ctl->qnt_vh = iq;
01209     sprintf(ctl->qnt_unit[iq], "m/s");
01210 } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
01211     ctl->qnt_vz = iq;
01212     sprintf(ctl->qnt_unit[iq], "m/s");
01213 } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
01214     ctl->qnt_pv = iq;
01215     sprintf(ctl->qnt_unit[iq], "PVU");
01216 } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
01217     ctl->qnt_tice = iq;
01218     sprintf(ctl->qnt_unit[iq], "K");
01219 } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
01220     ctl->qnt_tsts = iq;
01221     sprintf(ctl->qnt_unit[iq], "K");
01222 } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
01223     ctl->qnt_tnat = iq;
01224     sprintf(ctl->qnt_unit[iq], "K");
01225 } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
01226     ctl->qnt_stat = iq;
01227     sprintf(ctl->qnt_unit[iq], "-");
01228 } else
01229     scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
01230 }
01231
01232 /* Check quantity flags... */
01233 if (ctl->qnt_tsts >= 0)
01234     if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
01235         ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
01236
01237 /* Time steps of simulation... */
01238 ctl->direction =
01239     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
01240 if (ctl->direction != -1 && ctl->direction != 1)
01241     ERRMSG("Set DIRECTION to -1 or 1!");
01242 ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
01243 ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
01244
01245 /* Meteorological data... */
01246 ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
01247 ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
01248 ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
01249 ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
01250 ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
01251 ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
01252 ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
01253 ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
01254 if (ctl->met_np > EP)
01255     ERRMSG("Too many levels!");
01256 for (int ip = 0; ip < ctl->met_np; ip++)
01257     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
01258 ctl->met_tropo
01259     = (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "0", NULL);
01260 scan_ctl(filename, argc, argv, "MET_GEOPOT", -1, "-", ctl->met_geopot);
01261 scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
01262 ctl->met_dt_out =
01263     scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
01264
01265 /* Isosurface parameters... */
01266 ctl->isosurf
01267     = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
01268 scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);

```

```

01269
01270 /* Diffusion parameters... */
01271 ctl->turb_dx_trop
01272     = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
01273 ctl->turb_dx_strat
01274     = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
01275 ctl->turb_dz_trop
01276     = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
01277 ctl->turb_dz_strat
01278     = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
01279 ctl->turb_mesox =
01280     scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
01281 ctl->turb_mesoz =
01282     scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
01283
01284 /* Mass and life time... */
01285 ctl->molmass = scan_ctl(filename, argc, argv, "MOLMASS", -1, "1", NULL);
01286 ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
01287 ctl->tdec_strat =
01288     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
01289
01290 /* PSC analysis... */
01291 ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
01292 ctl->psc_hno3 =
01293     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
01294
01295 /* Output of atmospheric data... */
01296 scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
atm_basename);
01297 scan_ctl(filename, argc, argv, "ATM_GPFIL", -1, "-", ctl->atm_gpfile);
01298 ctl->atm_dt_out =
01299     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
01300 ctl->atm_filter =
01301     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
01302 ctl->atm_stride =
01303     (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
01304 ctl->atm_type =
01305     (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
01306
01307 /* Output of CSI data... */
01308 scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
csi_basename);
01309 ctl->csi_dt_out =
01310     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
01311 scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->
csi_obsfile);
01312 ctl->csi_obsmin =
01313     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
01314 ctl->csi_modmin =
01315     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
01316 ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
01317 ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
01318 ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
01319 ctl->csi_lon0 =
01320     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
01321 ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
01322 ctl->csi_nx =
01323     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
01324 ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
01325 ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
01326 ctl->csi_ny =
01327     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
01328
01329 /* Output of ensemble data... */
01330 scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
ens_basename);
01331
01332 /* Output of grid data... */
01333 scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
01334     ctl->grid_basename);
01335 scan_ctl(filename, argc, argv, "GRID_GPFIL", -1, "-", ctl->
grid_gpfile);
01336 ctl->grid_dt_out =
01337     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
01338 ctl->grid_sparse =
01339     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
01340 ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
01341 ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
01342 ctl->grid_nz =
01343     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
01344 ctl->grid_lon0 =
01345     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
01346 ctl->grid_lon1 =
01347     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
01348 ctl->grid_nx =
01349     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
01350 ctl->grid_lat0 =

```

```

01351     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
01352     ctl->grid_lat1 =
01353     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
01354     ctl->grid_ny =
01355     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
01356
01357     /* Output of profile data... */
01358     scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
01359     ctl->prof_basename);
01360     scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
prof_obsfile);
01361     ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
01362     ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
01363     ctl->prof_nz =
01364     (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
01365     ctl->prof_lon0 =
01366     scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
01367     ctl->prof_lon1 =
01368     scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
01369     ctl->prof_nx =
01370     (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
01371     ctl->prof_lat0 =
01372     scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
01373     ctl->prof_lat1 =
01374     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
01375     ctl->prof_ny =
01376     (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
01377
01378     /* Output of station data... */
01379     scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
01380     ctl->stat_basename);
01381     ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
01382     ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
01383     ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
01384 }

```

Here is the call graph for this function:



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

Read meteorological data file.

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

```

01391     {
01392
01393     char cmd[2 * LEN], levname[LEN], tstr[10];
01394
01395     float help[EX * EY];
01396
01397     int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
01398
01399     size_t np, nx, ny;
01400
01401     /* Write info... */
01402     printf("Read meteorological data: %s\n", filename);
01403
01404     /* Get time from filename... */
01405     sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
01406     year = atoi(tstr);
01407     sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
01408     mon = atoi(tstr);

```

```

01409     sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
01410     day = atoi(tstr);
01411     sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
01412     hour = atoi(tstr);
01413     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
01414
01415     /* Open netCDF file... */
01416     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
01417
01418         /* Try to stage meteo file... */
01419         if (ctl->met_stage[0] != '-') {
01420             sprintf(cmd, "%s %d %02d %02d %s", ctl->met_stage,
01421                 year, mon, day, hour, filename);
01422             if (system(cmd) != 0)
01423                 ERRMSG("Error while staging meteo data!");
01424         }
01425
01426         /* Try to open again... */
01427         if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
01428             WARN("File not found!");
01429             return 0;
01430         }
01431     }
01432
01433     /* Get dimensions... */
01434     NC(nc_inq_dimid(ncid, "lon", &dimid));
01435     NC(nc_inq_dimlen(ncid, dimid, &nx));
01436     if (nx < 2 || nx > EX)
01437         ERRMSG("Number of longitudes out of range!");
01438
01439     NC(nc_inq_dimid(ncid, "lat", &dimid));
01440     NC(nc_inq_dimlen(ncid, dimid, &ny));
01441     if (ny < 2 || ny > EY)
01442         ERRMSG("Number of latitudes out of range!");
01443
01444     sprintf(levname, "lev");
01445     NC(nc_inq_dimid(ncid, levname, &dimid));
01446     NC(nc_inq_dimlen(ncid, dimid, &np));
01447     if (np == 1) {
01448         sprintf(levname, "lev_2");
01449         NC(nc_inq_dimid(ncid, levname, &dimid));
01450         NC(nc_inq_dimlen(ncid, dimid, &np));
01451     }
01452     if (np < 2 || np > EP)
01453         ERRMSG("Number of levels out of range!");
01454
01455     /* Store dimensions... */
01456     met->np = (int) np;
01457     met->nx = (int) nx;
01458     met->ny = (int) ny;
01459
01460     /* Get horizontal grid... */
01461     NC(nc_inq_varid(ncid, "lon", &varid));
01462     NC(nc_get_var_double(ncid, varid, met->lon));
01463     NC(nc_inq_varid(ncid, "lat", &varid));
01464     NC(nc_get_var_double(ncid, varid, met->lat));
01465
01466     /* Read meteorological data... */
01467     read_met_help(ncid, "t", "T", met, met->t, 1.0);
01468     read_met_help(ncid, "u", "U", met, met->u, 1.0);
01469     read_met_help(ncid, "v", "V", met, met->v, 1.0);
01470     read_met_help(ncid, "w", "W", met, met->w, 0.01f);
01471     read_met_help(ncid, "q", "Q", met, met->h2o, (float) (MA / 18.01528));
01472     read_met_help(ncid, "o3", "O3", met, met->o3, (float) (MA / 48.00));
01473
01474     /* Meteo data on pressure levels... */
01475     if (ctl->met_np <= 0) {
01476
01477         /* Read pressure levels from file... */
01478         NC(nc_inq_varid(ncid, levname, &varid));
01479         NC(nc_get_var_double(ncid, varid, met->p));
01480         for (ip = 0; ip < met->np; ip++)
01481             met->p[ip] /= 100.;
01482
01483         /* Extrapolate data for lower boundary... */
01484         read_met_extrapolate(met);
01485     }
01486
01487     /* Meteo data on model levels... */
01488     else {
01489
01490         /* Read pressure data from file... */
01491         read_met_help(ncid, "pl", "PL", met, met->p, 0.01f);
01492
01493         /* Interpolate from model levels to pressure levels... */
01494         read_met_ml2pl(ctl, met, met->t);
01495         read_met_ml2pl(ctl, met, met->u);

```

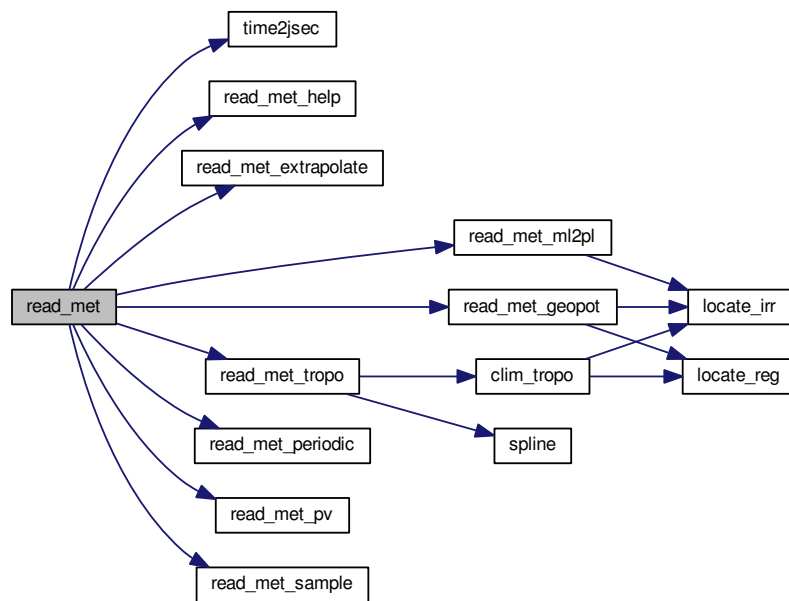


```

01496     read_met_ml2pl(ctl, met, met->v);
01497     read_met_ml2pl(ctl, met, met->w);
01498     read_met_ml2pl(ctl, met, met->h2o);
01499     read_met_ml2pl(ctl, met, met->o3);
01500
01501     /* Set pressure levels... */
01502     met->np = ctl->met_np;
01503     for (ip = 0; ip < met->np; ip++)
01504         met->p[ip] = ctl->met_p[ip];
01505 }
01506
01507 /* Check ordering of pressure levels... */
01508 for (ip = 1; ip < met->np; ip++)
01509     if (met->p[ip - 1] < met->p[ip])
01510         ERRMSG("Pressure levels must be descending!");
01511
01512 /* Read surface pressure... */
01513 if (nc_inq_varid(ncid, "ps", &varid) == NC_NOERR
01514     || nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
01515     NC(nc_get_var_float(ncid, varid, help));
01516     for (iy = 0; iy < met->ny; iy++)
01517         for (ix = 0; ix < met->nx; ix++)
01518             met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
01519 } else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR
01520     || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
01521     NC(nc_get_var_float(ncid, varid, help));
01522     for (iy = 0; iy < met->ny; iy++)
01523         for (ix = 0; ix < met->nx; ix++)
01524             met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
01525 } else
01526     for (ix = 0; ix < met->nx; ix++)
01527         for (iy = 0; iy < met->ny; iy++)
01528             met->ps[ix][iy] = met->p[0];
01529
01530 /* Create periodic boundary conditions... */
01531 read_met_periodic(met);
01532
01533 /* Calculate geopotential heights... */
01534 read_met_geopot(ctl, met);
01535
01536 /* Calculate potential vorticity... */
01537 read_met_pv(met);
01538
01539 /* Calculate tropopause pressure... */
01540 read_met_tropo(ctl, met);
01541
01542 /* Downsampling... */
01543 read_met_sample(ctl, met);
01544
01545 /* Close file... */
01546 NC(nc_close(ncid));
01547
01548 /* Return success... */
01549 return 1;
01550 }

```

Here is the call graph for this function:



5.21.2.20 void read_met_extrapolate (met_t * met)

Extrapolate meteorological data at lower boundary.

Definition at line 1554 of file libtrac.c.

```

01555         {
01556
01557     int ip, ip0, ix, iy;
01558
01559     /* Loop over columns... */
01560 #pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
01561     for (ix = 0; ix < met->nx; ix++)
01562         for (iy = 0; iy < met->ny; iy++) {
01563
01564             /* Find lowest valid data point... */
01565             for (ip0 = met->np - 1; ip0 >= 0; ip0--)
01566                 if (!gsl_finite(met->t[ix][iy][ip0])
01567                     || !gsl_finite(met->u[ix][iy][ip0])
01568                     || !gsl_finite(met->v[ix][iy][ip0])
01569                     || !gsl_finite(met->w[ix][iy][ip0]))
01570                     break;
01571
01572             /* Extrapolate... */
01573             for (ip = ip0; ip >= 0; ip--) {
01574                 met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
01575                 met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
01576                 met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
01577                 met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
01578                 met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
01579                 met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
01580             }
01581         }
01582     }
  
```

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

Calculate geopotential heights.

Definition at line 1586 of file libtrac.c.

```

01588         {
01589
01590     const int dx = 6, dy = 4;
01591
01592     static double logp[EP], topo_lat[EY], topo_lon[EX], topo_z[EX][EY];
01593
01594     static float help[EX][EY][EP];
01595
01596     static int init, topo_nx = -1, topo_ny;
01597
01598     FILE *in;
01599
01600     char line[LEN];
01601
01602     double lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;
01603
01604     int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;
01605
01606     /* Initialize geopotential heights... */
01607 #pragma omp parallel for default(shared) private(ix,iy,ip)
01608     for (ix = 0; ix < met->nx; ix++)
01609         for (iy = 0; iy < met->ny; iy++)
01610             for (ip = 0; ip < met->np; ip++)
01611                 met->z[ix][iy][ip] = GSL_NAN;
01612
01613     /* Check filename... */
01614     if (ctl->met_geopot[0] == '-')
01615         return;
01616
01617     /* Read surface geopotential... */
01618     if (!init) {
01619         init = 1;
01620
01621         /* Write info... */
01622         printf("Read surface geopotential: %s\n", ctl->met_geopot);
01623
01624         /* Open file... */
01625         if (!(in = fopen(ctl->met_geopot, "r")))
01626             ERRMSG("Cannot open file!");
01627
01628         /* Read data... */
01629         while (fgets(line, LEN, in))
01630             if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rz) == 3) {
01631                 if (rlon != rlon_old) {
01632                     if ((++topo_nx) > EX)
01633                         ERRMSG("Too many longitudes!");
01634                     topo_ny = 0;
01635                 }
01636                 rlon_old = rlon;
01637                 topo_lon[topo_nx] = rlon;
01638                 topo_lat[topo_ny] = rlat;
01639                 topo_z[topo_nx][topo_ny] = rz;
01640                 if ((++topo_ny) > EY)
01641                     ERRMSG("Too many latitudes!");
01642             }
01643         if ((++topo_nx) > EX)
01644             ERRMSG("Too many longitudes!");
01645
01646         /* Close file... */
01647         fclose(in);
01648
01649         /* Check grid spacing... */
01650         if (fabs(met->lon[0] - met->lon[1]) != fabs(topo_lon[0] - topo_lon[1])
01651             || fabs(met->lat[0] - met->lat[1]) != fabs(topo_lat[0] - topo_lat[1]))
01652             WARN("Grid spacing does not match!");
01653
01654         /* Calculate log pressure... */
01655         for (ip = 0; ip < met->np; ip++)
01656             logp[ip] = log(met->p[ip]);
01657     }
01658
01659     /* Apply hydrostatic equation to calculate geopotential heights... */
01660 #pragma omp parallel for default(shared) private(ix,iy,lon,lat,tx,ty,z0,z1,ip0,ts,ip)
01661     for (ix = 0; ix < met->nx; ix++) {
01662         /* Get longitude index... */
01663         lon = met->lon[ix];

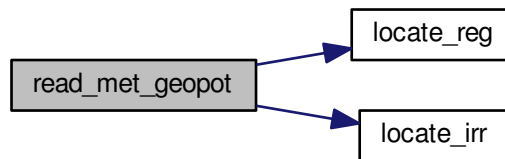
```

```

01665     if (lon < topo_lon[0])
01666         lon += 360;
01667     else if (lon > topo_lon[topo_nx - 1])
01668         lon -= 360;
01669     tx = locate_reg(topo_lon, topo_nx, lon);
01670
01671     /* Loop over latitudes... */
01672     for (iy = 0; iy < met->ny; iy++) {
01673
01674         /* Get latitude index... */
01675         lat = met->lat[iy];
01676         ty = locate_reg(topo_lat, topo_ny, lat);
01677
01678         /* Get surface height... */
01679         z0 = LIN(topo_lon[tx], topo_z[tx][ty],
01680                 topo_lon[tx + 1], topo_z[tx + 1][ty], lon);
01681         z1 = LIN(topo_lon[tx], topo_z[tx][ty + 1],
01682                 topo_lon[tx + 1], topo_z[tx + 1][ty + 1], lon);
01683         z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);
01684
01685         /* Find surface pressure level... */
01686         ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
01687
01688         /* Get surface data... */
01689         ts =
01690             LIN(met->p[ip0],
01691                 TVIRT(met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]),
01692                 met->p[ip0 + 1],
01693                 TVIRT(met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]),
01694                 met->ps[ix][iy]);
01695
01696         /* Upper part of profile... */
01697         met->z[ix][iy][ip0 + 1]
01698             = (float) (z0 + RI / MA / G0 * 0.5
01699                     * (ts + TVIRT(met->t[ix][iy][ip0 + 1],
01700                                   met->h2o[ix][iy][ip0 + 1]))
01701                     * (log(met->ps[ix][iy]) - logp[ip0 + 1]));
01702         for (ip = ip0 + 2; ip < met->np; ip++)
01703             met->z[ix][iy][ip]
01704                 = (float) (met->z[ix][iy][ip - 1] + RI / MA / G0 * 0.5 *
01705                         (TVIRT(met->t[ix][iy][ip - 1], met->h2o[ix][iy][ip - 1])
01706                          + TVIRT(met->t[ix][iy][ip], met->h2o[ix][iy][ip]))
01707                         * (logp[ip - 1] - logp[ip]));
01708     }
01709 }
01710
01711 /* Smoothing... */
01712 #pragma omp parallel for default(shared) private(ix,iy,ip,n,ix2,ix3,iy2)
01713 for (ix = 0; ix < met->nx; ix++)
01714     for (iy = 0; iy < met->ny; iy++)
01715         for (ip = 0; ip < met->np; ip++) {
01716             n = 0;
01717             help[ix][iy][ip] = 0;
01718             for (ix2 = ix - dx; ix2 <= ix + dx; ix2++) {
01719                 ix3 = ix2;
01720                 if (ix3 < 0)
01721                     ix3 += met->nx;
01722                 else if (ix3 >= met->nx)
01723                     ix3 -= met->nx;
01724                 for (iy2 = GSL_MAX(iy - dy, 0);
01725                     iy2 <= GSL_MIN(iy + dy, met->ny - 1); iy2++)
01726                     if (gsl_finite(met->z[ix3][iy2][ip])) {
01727                         help[ix][iy][ip] += met->z[ix3][iy2][ip];
01728                         n++;
01729                     }
01730             }
01731             if (n > 0)
01732                 help[ix][iy][ip] /= (float) n;
01733             else
01734                 help[ix][iy][ip] = GSL_NAN;
01735         }
01736
01737 /* Copy data... */
01738 #pragma omp parallel for default(shared) private(ix,iy,ip)
01739 for (ix = 0; ix < met->nx; ix++)
01740     for (iy = 0; iy < met->ny; iy++)
01741         for (ip = 0; ip < met->np; ip++)
01742             met->z[ix][iy][ip] = help[ix][iy][ip];
01743 }

```

Here is the call graph for this function:



5.21.2.22 void read_met_help (int *ncid*, char * *varname*, char * *varname2*, met_t * *met*, float *dest*[EX][EY][EP], float *scl*)

Read and convert variable from meteorological data file.

Definition at line 1747 of file libtrac.c.

```

01753     {
01754
01755     float *help;
01756
01757     int ip, ix, iy, varid;
01758
01759     /* Check if variable exists... */
01760     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
01761         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
01762             return;
01763
01764     /* Allocate... */
01765     ALLOC(help, float, EX * EY * EP);
01766
01767     /* Read data... */
01768     NC(nc_get_var_float(ncid, varid, help));
01769
01770     /* Copy and check data... */
01771     #pragma omp parallel for default(shared) private(ix,iy,ip)
01772     for (ix = 0; ix < met->nx; ix++)
01773         for (iy = 0; iy < met->ny; iy++)
01774             for (ip = 0; ip < met->np; ip++) {
01775                 dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
01776                 if (fabsf(dest[ix][iy][ip]) < 1e14f)
01777                     dest[ix][iy][ip] *= scl;
01778                 else
01779                     dest[ix][iy][ip] = GSL_NAN;
01780             }
01781
01782     /* Free... */
01783     free(help);
01784 }

```

5.21.2.23 void read_met_m2pl (ctl_t * *ctl*, met_t * *met*, float *var*[EX][EY][EP])

Convert meteorological data from model levels to pressure levels.

Definition at line 1788 of file libtrac.c.

```

01791     {
01792
01793     double aux[EP], p[EP], pt;
01794
01795     int ip, ip2, ix, iy;
01796

```

```

01797  /* Loop over columns... */
01798 #pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
01799 for (ix = 0; ix < met->nx; ix++)
01800     for (iy = 0; iy < met->ny; iy++) {
01801
01802         /* Copy pressure profile... */
01803         for (ip = 0; ip < met->np; ip++)
01804             p[ip] = met->p[ix][iy][ip];
01805
01806         /* Interpolate... */
01807         for (ip = 0; ip < ctl->met_np; ip++) {
01808             pt = ctl->met_p[ip];
01809             if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
01810                 pt = p[0];
01811             else if ((pt > p[met->np - 1] && p[1] > p[0])
01812                     || (pt < p[met->np - 1] && p[1] < p[0]))
01813                 pt = p[met->np - 1];
01814             ip2 = locate_irr(p, met->np, pt);
01815             aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
01816                          p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
01817         }
01818
01819         /* Copy data... */
01820         for (ip = 0; ip < ctl->met_np; ip++)
01821             var[ix][iy][ip] = (float) aux[ip];
01822     }
01823 }

```

Here is the call graph for this function:



5.21.2.24 void read_met_periodic(met_t * met)

Create meteorological data with periodic boundary conditions.

Definition at line 1827 of file libtrac.c.

```

01828     {
01829
01830         /* Check longitudes... */
01831         if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
01832                  + met->lon[1] - met->lon[0] - 360) < 0.01))
01833             return;
01834
01835         /* Increase longitude counter... */
01836         if ((++met->nx) > EX)
01837             ERRMSG("Cannot create periodic boundary conditions!");
01838
01839         /* Set longitude... */
01840         met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
lon[0];
01841
01842         /* Loop over latitudes and pressure levels... */
01843 #pragma omp parallel for default(shared)
01844         for (int iy = 0; iy < met->ny; iy++) {
01845             met->ps[met->nx - 1][iy] = met->ps[0][iy];
01846             met->pt[met->nx - 1][iy] = met->pt[0][iy];
01847             for (int ip = 0; ip < met->np; ip++) {
01848                 met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
01849                 met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
01850                 met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
01851                 met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
01852                 met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
01853                 met->pv[met->nx - 1][iy][ip] = met->pv[0][iy][ip];
01854                 met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
01855                 met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01856             }
01857         }
01858     }

```

5.21.2.25 void read_met_pv (met_t * met)

Calculate potential vorticity.

Definition at line 1862 of file libtrac.c.

```

01863         {
01864
01865         double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
01866             dtdp, dudp, dvdp, latr, vort, pows[EP];
01867
01868         int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
01869
01870         /* Set powers... */
01871         for (ip = 0; ip < met->np; ip++)
01872             pows[ip] = pow(1000. / met->p[ip], 0.286);
01873
01874         /* Loop over grid points... */
01875 #pragma omp parallel for default(shared)
01876         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)
01877         for (ix = 0; ix < met->nx; ix++) {
01878
01879             /* Set indices... */
01880             ix0 = GSL_MAX(ix - 1, 0);
01881             ix1 = GSL_MIN(ix + 1, met->nx - 1);
01882
01883             /* Loop over grid points... */
01884             for (iy = 0; iy < met->ny; iy++) {
01885
01886                 /* Set indices... */
01887                 iy0 = GSL_MAX(iy - 1, 0);
01888                 iy1 = GSL_MIN(iy + 1, met->ny - 1);
01889
01890                 /* Set auxiliary variables... */
01891                 latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
01892                 dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
01893                 dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
01894                 c0 = cos(met->lat[iy0] / 180. * M_PI);
01895                 c1 = cos(met->lat[iy1] / 180. * M_PI);
01896                 cr = cos(latr / 180. * M_PI);
01897                 vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
01898
01899                 /* Loop over grid points... */
01900                 for (ip = 0; ip < met->np; ip++) {
01901
01902                     /* Get gradients in longitude... */
01903                     dtdx = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
01904                     dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
01905
01906                     /* Get gradients in latitude... */
01907                     dtdy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
01908                     dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
01909
01910                     /* Set indices... */
01911                     ip0 = GSL_MAX(ip - 1, 0);
01912                     ip1 = GSL_MIN(ip + 1, met->np - 1);
01913
01914                     /* Get gradients in pressure... */
01915                     dp0 = 100. * (met->p[ip] - met->p[ip0]);
01916                     dp1 = 100. * (met->p[ip1] - met->p[ip]);
01917                     if (ip != ip0 && ip != ip1) {
01918                         denom = dp0 * dp1 * (dp0 + dp1);
01919                         dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
01920                             - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
01921                             + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
01922                             / denom;
01923                         dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
01924                             - dp1 * dp1 * met->u[ix][iy][ip0]
01925                             + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
01926                             / denom;
01927                         dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
01928                             - dp1 * dp1 * met->v[ix][iy][ip0]
01929                             + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
01930                             / denom;
01931                     } else {
01932                         denom = dp0 + dp1;
01933                         dtdp =
01934                             (met->t[ix][iy][ip1] * pows[ip1] -
01935                             met->t[ix][iy][ip0] * pows[ip0]) / denom;
01936                         dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
01937                         dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
01938                     }
01939                 }
01940             }
01941         }

```

```

01939         /* Calculate PV... */
01940         met->pv[ix][iy][ip] = (float)
01941             (1e6 * G0 *
01942             (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
01943     }
01944 }
01945 }
01946
01947 /* Fix for polar regions... */
01948 #pragma omp parallel for default(shared) private(ix,ip)
01949 for (ix = 0; ix < met->nx; ix++)
01950     for (ip = 0; ip < met->np; ip++) {
01951         met->pv[ix][0][ip]
01952             = met->pv[ix][1][ip]
01953             = met->pv[ix][2][ip];
01954         met->pv[ix][met->ny - 1][ip]
01955             = met->pv[ix][met->ny - 2][ip]
01956             = met->pv[ix][met->ny - 3][ip];
01957     }
01958 }

```

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

Downsampling of meteorological data.

Definition at line 1962 of file libtrac.c.

```

01964     {
01965
01966         met_t *help;
01967
01968         float w, wsum;
01969
01970         int ip, ip2, ix, ix2, ix3, iy, iy2;
01971
01972         /* Check parameters... */
01973         if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
01974             && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
01975             return;
01976
01977         /* Allocate... */
01978         ALLOC(help, met_t, 1);
01979
01980         /* Copy data... */
01981         help->nx = met->nx;
01982         help->ny = met->ny;
01983         help->np = met->np;
01984         memcpy(help->lon, met->lon, sizeof(met->lon));
01985         memcpy(help->lat, met->lat, sizeof(met->lat));
01986         memcpy(help->p, met->p, sizeof(met->p));
01987
01988         /* Smoothing... */
01989         for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
01990             for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
01991                 for (ip = 0; ip < met->np; ip += ctl->met_dp) {
01992                     help->ps[ix][iy] = 0;
01993                     help->pt[ix][iy] = 0;
01994                     help->z[ix][iy][ip] = 0;
01995                     help->t[ix][iy][ip] = 0;
01996                     help->u[ix][iy][ip] = 0;
01997                     help->v[ix][iy][ip] = 0;
01998                     help->w[ix][iy][ip] = 0;
01999                     help->pv[ix][iy][ip] = 0;
02000                     help->h2o[ix][iy][ip] = 0;
02001                     help->o3[ix][iy][ip] = 0;
02002                     wsum = 0;
02003                     for (ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1; ix2++) {
02004                         ix3 = ix2;
02005                         if (ix3 < 0)
02006                             ix3 += met->nx;
02007                         else if (ix3 >= met->nx)
02008                             ix3 -= met->nx;
02009
02010                         for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
02011                             iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
02012                             for (ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
02013                                 ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
02014                                 w = (float) (1.0 - fabs(ix - ix2) / ctl->met_sx)
02015                                     * (float) (1.0 - fabs(iy - iy2) / ctl->met_sy)
02016                                     * (float) (1.0 - fabs(ip - ip2) / ctl->met_sp);

```



```

02017         help->ps[ix][iy] += w * met->ps[ix3][iy2];
02018         help->pt[ix][iy] += w * met->pt[ix3][iy2];
02019         help->z[ix][iy][ip] += w * met->z[ix3][iy2][ip2];
02020         help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
02021         help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
02022         help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
02023         help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
02024         help->pv[ix][iy][ip] += w * met->pv[ix3][iy2][ip2];
02025         help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
02026         help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
02027         wsum += w;
02028     }
02029 }
02030 help->ps[ix][iy] /= wsum;
02031 help->pt[ix][iy] /= wsum;
02032 help->t[ix][iy][ip] /= wsum;
02033 help->z[ix][iy][ip] /= wsum;
02034 help->u[ix][iy][ip] /= wsum;
02035 help->v[ix][iy][ip] /= wsum;
02036 help->w[ix][iy][ip] /= wsum;
02037 help->pv[ix][iy][ip] /= wsum;
02038 help->h2o[ix][iy][ip] /= wsum;
02039 help->o3[ix][iy][ip] /= wsum;
02040 }
02041 }
02042 }
02043
02044 /* Downsampling... */
02045 met->nx = 0;
02046 for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
02047     met->lon[met->nx] = help->lon[ix];
02048     met->ny = 0;
02049     for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
02050         met->lat[met->ny] = help->lat[iy];
02051         met->ps[met->nx][met->ny] = help->ps[ix][iy];
02052         met->pt[met->nx][met->ny] = help->pt[ix][iy];
02053         met->np = 0;
02054         for (ip = 0; ip < help->np; ip += ctl->met_dp) {
02055             met->p[met->np] = help->p[ip];
02056             met->z[met->nx][met->ny][met->np] = help->z[ix][iy][ip];
02057             met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
02058             met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
02059             met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
02060             met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
02061             met->pv[met->nx][met->ny][met->np] = help->pv[ix][iy][ip];
02062             met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
02063             met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
02064             met->np++;
02065         }
02066         met->ny++;
02067     }
02068     met->nx++;
02069 }
02070
02071 /* Free... */
02072 free(help);
02073 }

```

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

Calculate tropopause pressure.

Definition at line 2077 of file libtrac.c.

```

02079     {
02080
02081         double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
02082             th2[200], z[EP], z2[200];
02083
02084         int found, ix, iy, iz, iz2;
02085
02086         /* Get altitude and pressure profiles... */
02087         for (iz = 0; iz < met->np; iz++)
02088             z[iz] = Z(met->p[iz]);
02089         for (iz = 0; iz <= 170; iz++) {
02090             z2[iz] = 4.5 + 0.1 * iz;
02091             p2[iz] = P(z2[iz]);
02092         }
02093
02094         /* Do not calculate tropopause... */

```

```

02095     if (ctl->met_tropo == 0)
02096         for (ix = 0; ix < met->nx; ix++)
02097             for (iy = 0; iy < met->ny; iy++)
02098                 met->pt[ix][iy] = GSL_NAN;
02099
02100     /* Use tropopause climatology... */
02101     else if (ctl->met_tropo == 1) {
02102 #pragma omp parallel for default(shared) private(ix,iy)
02103         for (ix = 0; ix < met->nx; ix++)
02104             for (iy = 0; iy < met->ny; iy++)
02105                 met->pt[ix][iy] = clim_tropo(met->time, met->lat[iy]);
02106     }
02107
02108     /* Use cold point... */
02109     else if (ctl->met_tropo == 2) {
02110
02111         /* Loop over grid points... */
02112 #pragma omp parallel for default(shared) private(ix,iy,iz,t,t2)
02113         for (ix = 0; ix < met->nx; ix++)
02114             for (iy = 0; iy < met->ny; iy++) {
02115
02116                 /* Interpolate temperature profile... */
02117                 for (iz = 0; iz < met->np; iz++)
02118                     t[iz] = met->t[ix][iy][iz];
02119                 spline(z, t, met->np, z2, t2, 171);
02120
02121                 /* Find minimum... */
02122                 iz = (int) gsl_stats_min_index(t2, 1, 171);
02123                 if (iz <= 0 || iz >= 170)
02124                     met->pt[ix][iy] = GSL_NAN;
02125                 else
02126                     met->pt[ix][iy] = p2[iz];
02127             }
02128     }
02129
02130     /* Use WMO definition... */
02131     else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
02132
02133         /* Loop over grid points... */
02134 #pragma omp parallel for default(shared) private(ix,iy,iz,iz2,t,t2,found)
02135         for (ix = 0; ix < met->nx; ix++)
02136             for (iy = 0; iy < met->ny; iy++) {
02137
02138                 /* Interpolate temperature profile... */
02139                 for (iz = 0; iz < met->np; iz++)
02140                     t[iz] = met->t[ix][iy][iz];
02141                 spline(z, t, met->np, z2, t2, 161);
02142
02143                 /* Find 1st tropopause... */
02144                 met->pt[ix][iy] = GSL_NAN;
02145                 for (iz = 0; iz <= 140; iz++) {
02146                     found = 1;
02147                     for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
02148                         if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
02149                             * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
02150                             found = 0;
02151                             break;
02152                         }
02153                     if (found) {
02154                         if (iz > 0 && iz < 140)
02155                             met->pt[ix][iy] = p2[iz];
02156                         break;
02157                     }
02158                 }
02159
02160                 /* Find 2nd tropopause... */
02161                 if (ctl->met_tropo == 4) {
02162                     met->pt[ix][iy] = GSL_NAN;
02163                     for (; iz <= 140; iz++) {
02164                         found = 1;
02165                         for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
02166                             if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
02167                                 * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) < 3.0) {
02168                                 found = 0;
02169                                 break;
02170                             }
02171                         if (found)
02172                             break;
02173                     }
02174                     for (; iz <= 140; iz++) {
02175                         found = 1;
02176                         for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
02177                             if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
02178                                 * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
02179                                 found = 0;
02180                                 break;
02181                             }

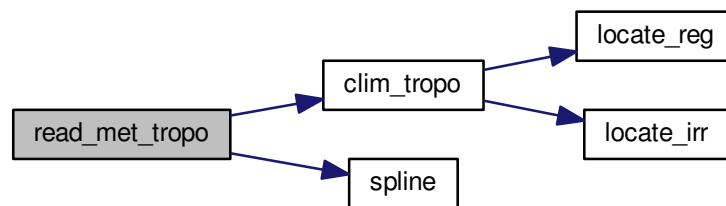
```

```

02182         if (found) {
02183             if (iz > 0 && iz < 140)
02184                 met->pt[ix][iy] = p2[iz];
02185             break;
02186         }
02187     }
02188 }
02189 }
02190 }
02191
02192 /* Use dynamical tropopause... */
02193 else if (ctl->met_tropo == 5) {
02194
02195     /* Loop over grid points... */
02196 #pragma omp parallel for default(shared) private(ix,iy,iz,pv,pv2,th,th2)
02197     for (ix = 0; ix < met->nx; ix++)
02198         for (iy = 0; iy < met->ny; iy++) {
02199
02200             /* Interpolate potential vorticity profile... */
02201             for (iz = 0; iz < met->np; iz++)
02202                 pv[iz] = met->pv[ix][iy][iz];
02203             spline(z, pv, met->np, z2, pv2, 161);
02204
02205             /* Interpolate potential temperature profile... */
02206             for (iz = 0; iz < met->np; iz++)
02207                 th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
02208             spline(z, th, met->np, z2, th2, 161);
02209
02210             /* Find dynamical tropopause 3.5 PVU + 380 K */
02211             met->pt[ix][iy] = GSL_NAN;
02212             for (iz = 0; iz <= 160; iz++)
02213                 if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
02214                     if (iz > 0 && iz < 160)
02215                         met->pt[ix][iy] = p2[iz];
02216                     break;
02217                 }
02218         }
02219     }
02220
02221 else
02222     ERRMSG("Cannot calculate tropopause!");
02223 }

```

Here is the call graph for this function:



5.21.2.28 `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 2227 of file libtrac.c.

```

02234     {
02235
02236     FILE *in = NULL;

```

```

02237
02238 char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
02239      msg[2 * LEN], rvarname[LEN], rval[LEN];
02240
02241 int contain = 0, i;
02242
02243 /* Open file... */
02244 if (filename[strlen(filename) - 1] != '-')
02245     if (!(in = fopen(filename, "r")))
02246         ERRMSG("Cannot open file!");
02247
02248 /* Set full variable name... */
02249 if (arridx >= 0) {
02250     sprintf(fullname1, "%s[%d]", varname, arridx);
02251     sprintf(fullname2, "%s[*]", varname);
02252 } else {
02253     sprintf(fullname1, "%s", varname);
02254     sprintf(fullname2, "%s", varname);
02255 }
02256
02257 /* Read data... */
02258 if (in != NULL)
02259     while (fgets(line, LEN, in))
02260         if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
02261             if (strcasemp(rvarname, fullname1) == 0 ||
02262                 strcasemp(rvarname, fullname2) == 0) {
02263                 contain = 1;
02264                 break;
02265             }
02266 for (i = 1; i < argc - 1; i++)
02267     if (strcasemp(argv[i], fullname1) == 0 ||
02268         strcasemp(argv[i], fullname2) == 0) {
02269         sprintf(rval, "%s", argv[i + 1]);
02270         contain = 1;
02271         break;
02272     }
02273
02274 /* Close file... */
02275 if (in != NULL)
02276     fclose(in);
02277
02278 /* Check for missing variables... */
02279 if (!contain) {
02280     if (strlen(defvalue) > 0)
02281         sprintf(rval, "%s", defvalue);
02282     else {
02283         sprintf(msg, "Missing variable %s!\n", fullname1);
02284         ERRMSG(msg);
02285     }
02286 }
02287
02288 /* Write info... */
02289 printf("%s = %s\n", fullname1, rval);
02290
02291 /* Return values... */
02292 if (value != NULL)
02293     sprintf(value, "%s", rval);
02294 return atof(rval);
02295 }

```

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

Spline interpolation.

Definition at line 2299 of file libtrac.c.

```

02305     {
02306
02307     gsl_interp_accel *acc;
02308
02309     gsl_spline *s;
02310
02311     /* Allocate... */
02312     acc = gsl_interp_accel_alloc();
02313     s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
02314
02315     /* Interpolate temperature profile... */
02316     gsl_spline_init(s, x, y, (size_t) n);
02317     for (int i = 0; i < n2; i++)
02318         y2[i] = gsl_spline_eval(s, x2[i], acc);

```

```

02319
02320  /* Free... */
02321  gsl_spline_free(s);
02322  gsl_interp_accel_free(acc);
02323 }

```

5.21.2.30 double stddev (double * *data*, int *n*)

Calculate standard deviation.

Definition at line 2327 of file [libtrac.c](#).

```

02329      {
02330
02331      if (n <= 0)
02332          return 0;
02333
02334      double avg = 0, rms = 0;
02335
02336      for (int i = 0; i < n; ++i)
02337          avg += data[i];
02338      avg /= n;
02339
02340      for (int i = 0; i < n; ++i)
02341          rms += SQR(data[i] - avg);
02342
02343      return sqrt(rms / (n - 1));
02344 }

```

5.21.2.31 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 2348 of file [libtrac.c](#).

```

02356      {
02357
02358      struct tm t0, t1;
02359
02360      t0.tm_year = 100;
02361      t0.tm_mon = 0;
02362      t0.tm_mday = 1;
02363      t0.tm_hour = 0;
02364      t0.tm_min = 0;
02365      t0.tm_sec = 0;
02366
02367      t1.tm_year = year - 1900;
02368      t1.tm_mon = mon - 1;
02369      t1.tm_mday = day;
02370      t1.tm_hour = hour;
02371      t1.tm_min = min;
02372      t1.tm_sec = sec;
02373
02374      *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
02375 }

```

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

Measure wall-clock time.

Definition at line 2379 of file [libtrac.c](#).

```

02382         {
02383
02384     static double starttime[NTIMER], runtime[NTIMER];
02385
02386     /* Check id... */
02387     if (id < 0 || id >= NTIMER)
02388         ERRMSG("Too many timers!");
02389
02390     /* Start timer... */
02391     if (mode == 1) {
02392         if (starttime[id] <= 0)
02393             starttime[id] = omp_get_wtime();
02394         else
02395             ERRMSG("Timer already started!");
02396     }
02397
02398     /* Stop timer... */
02399     else if (mode == 2) {
02400         if (starttime[id] > 0) {
02401             runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
02402             starttime[id] = -1;
02403         }
02404     }
02405
02406     /* Print timer... */
02407     else if (mode == 3) {
02408         printf("%s = %.3f s\n", name, runtime[id]);
02409         runtime[id] = 0;
02410     }
02411 }

```

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

Write atmospheric data.

Definition at line 2415 of file libtrac.c.

```

02419         {
02420
02421     FILE *in, *out;
02422
02423     char line[LEN];
02424
02425     double r, t0, t1;
02426
02427     int ip, iq, year, mon, day, hour, min, sec;
02428
02429     /* Set time interval for output... */
02430     t0 = t - 0.5 * ctl->dt_mod;
02431     t1 = t + 0.5 * ctl->dt_mod;
02432
02433     /* Write info... */
02434     printf("Write atmospheric data: %s\n", filename);
02435
02436     /* Write ASCII data... */
02437     if (ctl->atm_type == 0) {
02438
02439         /* Check if gnuplot output is requested... */
02440         if (ctl->atm_gpfile[0] != '-') {
02441
02442             /* Create gnuplot pipe... */
02443             if (!(out = popen("gnuplot", "w")))
02444                 ERRMSG("Cannot create pipe to gnuplot!");
02445
02446             /* Set plot filename... */
02447             fprintf(out, "set out \"%s.png\"\n", filename);
02448
02449             /* Set time string... */
02450             jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02451             fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02452                     year, mon, day, hour, min);
02453
02454             /* Dump gnuplot file to pipe... */
02455             if (!(in = fopen(ctl->atm_gpfile, "r")))
02456                 ERRMSG("Cannot open file!");
02457             while (fgets(line, LEN, in))
02458                 fprintf(out, "%s", line);
02459             fclose(in);
02460         }
02461     }

```

```

02462     else {
02463
02464         /* Create file... */
02465         if (!(out = fopen(filename, "w")))
02466             ERRMSG("Cannot create file!");
02467     }
02468
02469     /* Write header... */
02470     fprintf(out,
02471             "# $1 = time [s]\n"
02472             "# $2 = altitude [km]\n"
02473             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02474     for (iq = 0; iq < ctl->nq; iq++)
02475         fprintf(out, "# $i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
02476                 ctl->qnt_unit[iq]);
02477     fprintf(out, "\n");
02478
02479     /* Write data... */
02480     for (ip = 0; ip < atm->np; ip += ctl->atm_stride) {
02481
02482         /* Check time... */
02483         if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
02484             continue;
02485
02486         /* Write output... */
02487         fprintf(out, "%.2f %g %g", atm->time[ip], Z(atm->p[ip]),
02488                 atm->lon[ip], atm->lat[ip]);
02489         for (iq = 0; iq < ctl->nq; iq++) {
02490             fprintf(out, " ");
02491             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02492         }
02493         fprintf(out, "\n");
02494     }
02495
02496     /* Close file... */
02497     fclose(out);
02498 }
02499
02500 /* Write binary data... */
02501 else if (ctl->atm_type == 1) {
02502
02503     /* Create file... */
02504     if (!(out = fopen(filename, "w")))
02505         ERRMSG("Cannot create file!");
02506
02507     /* Write data... */
02508     FWRITE(&atm->np, int,
02509           1,
02510           out);
02511     FWRITE(atm->time, double,
02512           (size_t) atm->np,
02513           out);
02514     FWRITE(atm->p, double,
02515           (size_t) atm->np,
02516           out);
02517     FWRITE(atm->lon, double,
02518           (size_t) atm->np,
02519           out);
02520     FWRITE(atm->lat, double,
02521           (size_t) atm->np,
02522           out);
02523     for (iq = 0; iq < ctl->nq; iq++)
02524         FWRITE(atm->q[iq], double,
02525               (size_t) atm->np,
02526               out);
02527
02528     /* Close file... */
02529     fclose(out);
02530 }
02531
02532 /* Error... */
02533 else
02534     ERRMSG("Atmospheric data type not supported!");
02535 }

```

Here is the call graph for this function:



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

Write CSI data.

Definition at line 2539 of file libtrac.c.

```

02543     {
02544
02545     static FILE *in, *out;
02546
02547     static char line[LEN];
02548
02549     static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
02550         rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
02551
02552     static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
02553
02554     /* Init... */
02555     if (t == ctl->t_start) {
02556
02557         /* Check quantity index for mass... */
02558         if (ctl->qnt_m < 0)
02559             ERRMSG("Need quantity mass!");
02560
02561         /* Open observation data file... */
02562         printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
02563         if (!(in = fopen(ctl->csi_obsfile, "r")))
02564             ERRMSG("Cannot open file!");
02565
02566         /* Create new file... */
02567         printf("Write CSI data: %s\n", filename);
02568         if (!(out = fopen(filename, "w")))
02569             ERRMSG("Cannot create file!");
02570
02571         /* Write header... */
02572         fprintf(out,
02573             "# $1 = time [s]\n"
02574             "# $2 = number of hits (cx)\n"
02575             "# $3 = number of misses (cy)\n"
02576             "# $4 = number of false alarms (cz)\n"
02577             "# $5 = number of observations (cx + cy)\n"
02578             "# $6 = number of forecasts (cx + cz)\n"
02579             "# $7 = bias (forecasts/observations) [%%]\n"
02580             "# $8 = probability of detection (POD) [%%]\n"
02581             "# $9 = false alarm rate (FAR) [%%]\n"
02582             "# $10 = critical success index (CSI) [%%]\n\n");
02583     }
02584
02585     /* Set time interval... */
02586     t0 = t - 0.5 * ctl->dt_mod;
02587     t1 = t + 0.5 * ctl->dt_mod;
02588
02589     /* Initialize grid cells... */
02590     #pragma omp parallel for default(shared) private(ix,iy,iz)
02591     for (ix = 0; ix < ctl->csi_nx; ix++)
02592         for (iy = 0; iy < ctl->csi_ny; iy++)
02593             for (iz = 0; iz < ctl->csi_nz; iz++)
02594                 modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
02595
02596     /* Read observation data... */
02597     while (fgets(line, LEN, in)) {
02598
02599         /* Read data... */
  
```



```

02600     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &rln, &rln) !=
02601         5)
02602         continue;
02603
02604     /* Check time... */
02605     if (rt < t0)
02606         continue;
02607     if (rt > t1)
02608         break;
02609
02610     /* Calculate indices... */
02611     ix = (int) ((rln - ctl->csi_lon0)
02612                / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02613     iy = (int) ((rln - ctl->csi_lat0)
02614                / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02615     iz = (int) ((rz - ctl->csi_z0)
02616                / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02617
02618     /* Check indices... */
02619     if (ix < 0 || ix >= ctl->csi_nx ||
02620         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02621         continue;
02622
02623     /* Get mean observation index... */
02624     obsmean[ix][iy][iz] += robs;
02625     obscount[ix][iy][iz]++;
02626 }
02627
02628 /* Analyze model data... */
02629 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
02630 for (ip = 0; ip < atm->np; ip++) {
02631
02632     /* Check time... */
02633     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02634         continue;
02635
02636     /* Get indices... */
02637     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
02638                / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02639     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
02640                / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02641     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
02642                / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02643
02644     /* Check indices... */
02645     if (ix < 0 || ix >= ctl->csi_nx ||
02646         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02647         continue;
02648
02649     /* Get total mass in grid cell... */
02650     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02651 }
02652
02653 /* Analyze all grid cells... */
02654 #pragma omp parallel for default(shared) private(ix,iy,iz,dlon,dlat,lat,area)
02655 for (ix = 0; ix < ctl->csi_nx; ix++)
02656     for (iy = 0; iy < ctl->csi_ny; iy++)
02657         for (iz = 0; iz < ctl->csi_nz; iz++) {
02658
02659             /* Calculate mean observation index... */
02660             if (obscount[ix][iy][iz] > 0)
02661                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
02662
02663             /* Calculate column density... */
02664             if (modmean[ix][iy][iz] > 0) {
02665                 dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
02666                 dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
02667                 lat = ctl->csi_lat0 + dlat * (iy + 0.5);
02668                 area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
02669                     * cos(lat * M_PI / 180.);
02670                 modmean[ix][iy][iz] /= (1e6 * area);
02671             }
02672
02673             /* Calculate CSI... */
02674             if (obscount[ix][iy][iz] > 0) {
02675                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02676                     modmean[ix][iy][iz] >= ctl->csi_modmin)
02677                     cx++;
02678                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02679                     modmean[ix][iy][iz] < ctl->csi_modmin)
02680                     cy++;
02681                 else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
02682                     modmean[ix][iy][iz] >= ctl->csi_modmin)
02683                     cz++;
02684             }
02685         }
02686

```

```

02687  /* Write output... */
02688  if (fmod(t, ctl->csi_dt_out) == 0) {
02689
02690      /* Write... */
02691      fprintf(out, "%.2f %d %d %d %d %d %g %g %g\n",
02692              t, cx, cy, cz, cx + cy, cx + cz,
02693              (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
02694              (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
02695              (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
02696              (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
02697
02698      /* Set counters to zero... */
02699      cx = cy = cz = 0;
02700  }
02701
02702  /* Close file... */
02703  if (t == ctl->t_stop)
02704      fclose(out);
02705  }

```

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

Write ensemble data.

Definition at line 2709 of file libtrac.c.

```

02713      {
02714
02715      static FILE *out;
02716
02717      static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
02718          t0, t1, x[NENS][3], xm[3];
02719
02720      static int ip, iq;
02721
02722      static size_t i, n;
02723
02724      /* Init... */
02725      if (t == ctl->t_start) {
02726
02727          /* Check quantities... */
02728          if (ctl->qnt_ens < 0)
02729              ERRMSG("Missing ensemble IDs!");
02730
02731          /* Create new file... */
02732          printf("Write ensemble data: %s\n", filename);
02733          if (!(out = fopen(filename, "w")))
02734              ERRMSG("Cannot create file!");
02735
02736          /* Write header... */
02737          fprintf(out,
02738                  "# $1 = time [s]\n"
02739                  "# $2 = altitude [km]\n"
02740                  "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02741          for (iq = 0; iq < ctl->nq; iq++)
02742              fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
02743                      ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02744          for (iq = 0; iq < ctl->nq; iq++)
02745              fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
02746                      ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02747          fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
02748      }
02749
02750      /* Set time interval... */
02751      t0 = t - 0.5 * ctl->dt_mod;
02752      t1 = t + 0.5 * ctl->dt_mod;
02753
02754      /* Init... */
02755      ens = GSL_NAN;
02756      n = 0;
02757
02758      /* Loop over air parcels... */
02759      for (ip = 0; ip < atm->np; ip++) {
02760
02761          /* Check time... */
02762          if (atm->time[ip] < t0 || atm->time[ip] > t1)
02763              continue;
02764
02765          /* Check ensemble id... */
02766          if (atm->q[ctl->qnt_ens][ip] != ens) {

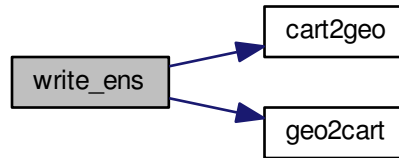
```

```

02767
02768     /* Write results... */
02769     if (n > 0) {
02770
02771         /* Get mean position... */
02772         xm[0] = xm[1] = xm[2] = 0;
02773         for (i = 0; i < n; i++) {
02774             xm[0] += x[i][0] / (double) n;
02775             xm[1] += x[i][1] / (double) n;
02776             xm[2] += x[i][2] / (double) n;
02777         }
02778         cart2geo(xm, &dummy, &lon, &lat);
02779         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
02780             lat);
02781
02782         /* Get quantity statistics... */
02783         for (iq = 0; iq < ctl->nq; iq++) {
02784             fprintf(out, " ");
02785             fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02786         }
02787         for (iq = 0; iq < ctl->nq; iq++) {
02788             fprintf(out, " ");
02789             fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02790         }
02791         fprintf(out, " %lu\n", n);
02792     }
02793
02794     /* Init new ensemble... */
02795     ens = atm->q[ctl->qnt_ens][ip];
02796     n = 0;
02797 }
02798
02799 /* Save data... */
02800 p[n] = atm->p[ip];
02801 geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
02802 for (iq = 0; iq < ctl->nq; iq++)
02803     q[iq][n] = atm->q[iq][ip];
02804 if ((++n) >= NENS)
02805     ERRMSG("Too many data points!");
02806 }
02807
02808 /* Write results... */
02809 if (n > 0) {
02810
02811     /* Get mean position... */
02812     xm[0] = xm[1] = xm[2] = 0;
02813     for (i = 0; i < n; i++) {
02814         xm[0] += x[i][0] / (double) n;
02815         xm[1] += x[i][1] / (double) n;
02816         xm[2] += x[i][2] / (double) n;
02817     }
02818     cart2geo(xm, &dummy, &lon, &lat);
02819     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
02820
02821     /* Get quantity statistics... */
02822     for (iq = 0; iq < ctl->nq; iq++) {
02823         fprintf(out, " ");
02824         fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02825     }
02826     for (iq = 0; iq < ctl->nq; iq++) {
02827         fprintf(out, " ");
02828         fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02829     }
02830     fprintf(out, " %lu\n", n);
02831 }
02832
02833 /* Close file... */
02834 if (t == ctl->t_stop)
02835     fclose(out);
02836 }

```

Here is the call graph for this function:



5.21.2.36 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 2840 of file libtrac.c.

```

02846     {
02847
02848     FILE *in, *out;
02849
02850     char line[LEN];
02851
02852     static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
02853       area, rho_air, press, temp, cd, vmr, t0, t1, r;
02854
02855     static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec;
02856
02857     /* Check dimensions... */
02858     if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
02859       ERRMSG("Grid dimensions too large!");
02860
02861     /* Set time interval for output... */
02862     t0 = t - 0.5 * ctl->dt_mod;
02863     t1 = t + 0.5 * ctl->dt_mod;
02864
02865     /* Set grid box size... */
02866     dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
02867     dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
02868     dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
02869
02870     /* Initialize grid... */
02871     #pragma omp parallel for default(shared) private(ix,iy,iz)
02872     for (ix = 0; ix < ctl->grid_nx; ix++)
02873       for (iy = 0; iy < ctl->grid_ny; iy++)
02874         for (iz = 0; iz < ctl->grid_nz; iz++) {
02875           mass[ix][iy][iz] = 0;
02876           np[ix][iy][iz] = 0;
02877         }
02878
02879     /* Average data... */
02880     #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
02881     for (ip = 0; ip < atm->np; ip++)
02882       if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
02883
02884         /* Get index... */
02885         ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
02886         iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
02887         iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
02888
02889         /* Check indices... */
02890         if (ix < 0 || ix >= ctl->grid_nx ||
02891             iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
02892           continue;
02893
02894         /* Add mass... */
02895         if (ctl->qnt_m >= 0)
02896           mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02897         np[ix][iy][iz]++;
02898       }
02899   }

```

```

02898     }
02899
02900     /* Check if gnuplot output is requested... */
02901     if (ctl->grid_gpfile[0] != '-') {
02902
02903         /* Write info... */
02904         printf("Plot grid data: %s.png\n", filename);
02905
02906         /* Create gnuplot pipe... */
02907         if (!(out = popen("gnuplot", "w")))
02908             ERRMSG("Cannot create pipe to gnuplot!");
02909
02910         /* Set plot filename... */
02911         fprintf(out, "set out \"%s.png\"\n", filename);
02912
02913         /* Set time string... */
02914         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02915         fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02916             year, mon, day, hour, min);
02917
02918         /* Dump gnuplot file to pipe... */
02919         if (!(in = fopen(ctl->grid_gpfile, "r")))
02920             ERRMSG("Cannot open file!");
02921         while (fgets(line, LEN, in))
02922             fprintf(out, "%s", line);
02923         fclose(in);
02924     }
02925
02926     else {
02927
02928         /* Write info... */
02929         printf("Write grid data: %s\n", filename);
02930
02931         /* Create file... */
02932         if (!(out = fopen(filename, "w")))
02933             ERRMSG("Cannot create file!");
02934     }
02935
02936     /* Write header... */
02937     fprintf(out,
02938         "# $1 = time [s]\n"
02939         "# $2 = altitude [km]\n"
02940         "# $3 = longitude [deg]\n"
02941         "# $4 = latitude [deg]\n"
02942         "# $5 = surface area [km^2]\n"
02943         "# $6 = layer width [km]\n"
02944         "# $7 = number of particles [l]\n"
02945         "# $8 = column density [kg/m^2]\n"
02946         "# $9 = volume mixing ratio [l]\n\n");
02947
02948     /* Write data... */
02949     for (ix = 0; ix < ctl->grid_nx; ix++) {
02950         if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
02951             fprintf(out, "\n");
02952         for (iy = 0; iy < ctl->grid_ny; iy++) {
02953             if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
02954                 fprintf(out, "\n");
02955             for (iz = 0; iz < ctl->grid_nz; iz++)
02956                 if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
02957
02958                     /* Set coordinates... */
02959                     z = ctl->grid_z0 + dz * (iz + 0.5);
02960                     lon = ctl->grid_lon0 + dlon * (ix + 0.5);
02961                     lat = ctl->grid_lat0 + dlat * (iy + 0.5);
02962
02963                     /* Get pressure and temperature... */
02964                     press = P(z);
02965                     intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
02966                         NULL, &temp, NULL, NULL, NULL, NULL, NULL);
02967
02968                     /* Calculate surface area... */
02969                     area = dlat * dlon * SQR(RE * M_PI / 180.)
02970                         * cos(lat * M_PI / 180.);
02971
02972                     /* Calculate column density... */
02973                     cd = mass[ix][iy][iz] / (1e6 * area);
02974
02975                     /* Calculate volume mixing ratio... */
02976                     rho_air = 100. * press / (RA * temp);
02977                     vmr = MA / ctl->molmass * mass[ix][iy][iz]
02978                         / (rho_air * 1e6 * area * 1e3 * dz);
02979
02980                     /* Write output... */
02981                     fprintf(out, "%.2f %g %g %g %g %d %g %g\n",
02982                         t, z, lon, lat, area, dz, np[ix][iy][iz], cd, vmr);
02983                 }
02984             }
02985         }
02986     }

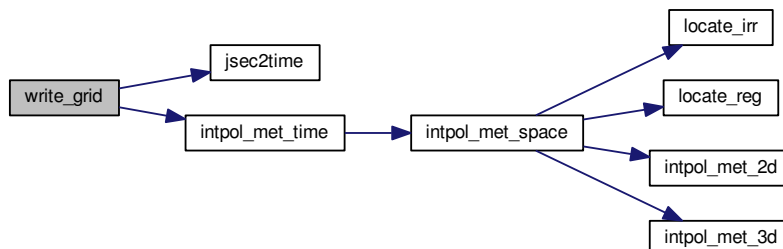
```

```

02985     }
02986
02987     /* Close file... */
02988     fclose(out);
02989 }

```

Here is the call graph for this function:



5.21.2.37 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 2993 of file libtrac.c.

```

02999     {
03000
03001     static FILE *in, *out;
03002
03003     static char line[LEN];
03004
03005     static double mass[GX][GY][GZ], obsmean[GX][GY], obsmean2[GX][GY], rt, rz,
03006         rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp,
03007         rho_air, vmr, h2o, o3;
03008
03009     static int obscount[GX][GY], ip, ix, iy, iz, okay;
03010
03011     /* Init... */
03012     if (t == ctl->t_start) {
03013
03014         /* Check quantity index for mass... */
03015         if (ctl->qnt_m < 0)
03016             ERRMSG("Need quantity mass!");
03017
03018         /* Check dimensions... */
03019         if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
03020             ERRMSG("Grid dimensions too large!");
03021
03022         /* Open observation data file... */
03023         printf("Read profile observation data: %s\n", ctl->prof_obsfile);
03024         if (!(in = fopen(ctl->prof_obsfile, "r")))
03025             ERRMSG("Cannot open file!");
03026
03027         /* Create new output file... */
03028         printf("Write profile data: %s\n", filename);
03029         if (!(out = fopen(filename, "w")))
03030             ERRMSG("Cannot create file!");
03031
03032         /* Write header... */
03033         fprintf(out,
03034             "# $1 = time [s]\n"
03035             "# $2 = altitude [km]\n"
03036             "# $3 = longitude [deg]\n"
03037             "# $4 = latitude [deg]\n"
03038             "# $5 = pressure [hPa]\n"
03039             "# $6 = temperature [K]\n"
03040             "# $7 = volume mixing ratio [1]\n"

```

```

03041         "# $8 = H2O volume mixing ratio [1]\n"
03042         "# $9 = O3 volume mixing ratio [1]\n"
03043         "# $10 = observed BT index (mean) [K]\n"
03044         "# $11 = observed BT index (sigma) [K]\n");
03045
03046     /* Set grid box size... */
03047     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
03048     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
03049     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
03050 }
03051
03052 /* Set time interval... */
03053 t0 = t - 0.5 * ctl->dt_mod;
03054 t1 = t + 0.5 * ctl->dt_mod;
03055
03056 /* Initialize... */
03057 #pragma omp parallel for default(shared) private(ix,iy,iz)
03058 for (ix = 0; ix < ctl->prof_nx; ix++)
03059     for (iy = 0; iy < ctl->prof_ny; iy++) {
03060         obsmean[ix][iy] = 0;
03061         obsmean2[ix][iy] = 0;
03062         obscount[ix][iy] = 0;
03063         for (iz = 0; iz < ctl->prof_nz; iz++)
03064             mass[ix][iy][iz] = 0;
03065     }
03066
03067 /* Read observation data... */
03068 while (fgets(line, LEN, in)) {
03069
03070     /* Read data... */
03071     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &robs) !=
03072         5)
03073         continue;
03074
03075     /* Check time... */
03076     if (rt < t0)
03077         continue;
03078     if (rt > t1)
03079         break;
03080
03081     /* Calculate indices... */
03082     ix = (int) ((rln - ctl->prof_lon0) / dlon);
03083     iy = (int) ((rln - ctl->prof_lat0) / dlat);
03084
03085     /* Check indices... */
03086     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
03087         continue;
03088
03089     /* Get mean observation index... */
03090     obsmean[ix][iy] += robs;
03091     obsmean2[ix][iy] += SQR(robs);
03092     obscount[ix][iy]++;
03093 }
03094
03095 /* Analyze model data... */
03096 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
03097 for (ip = 0; ip < atm->np; ip++) {
03098
03099     /* Check time... */
03100     if (atm->time[ip] < t0 || atm->time[ip] > t1)
03101         continue;
03102
03103     /* Get indices... */
03104     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
03105     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
03106     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
03107
03108     /* Check indices... */
03109     if (ix < 0 || ix >= ctl->prof_nx ||
03110         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
03111         continue;
03112
03113     /* Get total mass in grid cell... */
03114     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
03115 }
03116
03117 /* Extract profiles... */
03118 for (ix = 0; ix < ctl->prof_nx; ix++)
03119     for (iy = 0; iy < ctl->prof_ny; iy++)
03120         if (obscount[ix][iy] > 0) {
03121
03122             /* Check profile... */
03123             okay = 0;
03124             for (iz = 0; iz < ctl->prof_nz; iz++)
03125                 if (mass[ix][iy][iz] > 0) {
03126                     okay = 1;
03127                     break;

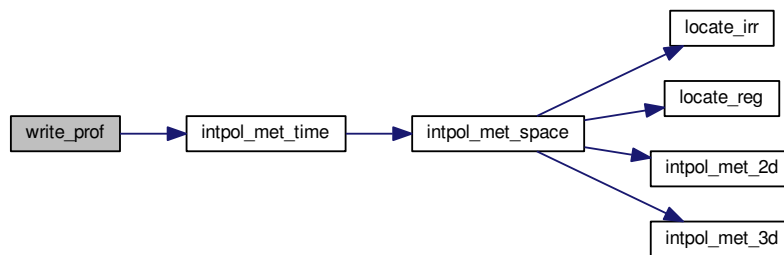
```

```

03128     }
03129     if (!okay)
03130         continue;
03131
03132     /* Write output... */
03133     fprintf(out, "\n");
03134
03135     /* Loop over altitudes... */
03136     for (iz = 0; iz < ctl->prof_nz; iz++) {
03137
03138         /* Set coordinates... */
03139         z = ctl->prof_z0 + dz * (iz + 0.5);
03140         lon = ctl->prof_lon0 + dlon * (ix + 0.5);
03141         lat = ctl->prof_lat0 + dlat * (iy + 0.5);
03142
03143         /* Get pressure and temperature... */
03144         press = P(z);
03145         intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
03146                       NULL, &temp, NULL, NULL, NULL, NULL, &h2o, &o3);
03147
03148         /* Calculate surface area... */
03149         area = dlat * dlon * SQR(M_PI * RE / 180.)
03150              * cos(lat * M_PI / 180.);
03151
03152         /* Calculate volume mixing ratio... */
03153         rho_air = 100. * press / (RA * temp);
03154         vmr = MA / ctl->molmass * mass[ix][iy][iz]
03155             / (rho_air * area * dz * 1e9);
03156
03157         /* Write output... */
03158         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
03159               t, z, lon, lat, press, temp, vmr, h2o, o3,
03160               obsmean[ix][iy] / obscount[ix][iy],
03161               sqrt(obsmean2[ix][iy] / obscount[ix][iy]
03162                   - SQR(obsmean[ix][iy] / obscount[ix][iy])));
03163     }
03164 }
03165
03166 /* Close file... */
03167 if (t == ctl->t_stop)
03168     fclose(out);
03169 }

```

Here is the call graph for this function:



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

Write station data.

Definition at line 3173 of file libtrac.c.

```

03177     {
03178
03179     static FILE *out;
03180
03181     static double rmax2, t0, t1, x0[3], x1[3];

```



```

03182
03183 /* Init... */
03184 if (t == ctl->t_start) {
03185
03186     /* Write info... */
03187     printf("Write station data: %s\n", filename);
03188
03189     /* Create new file... */
03190     if (!(out = fopen(filename, "w")))
03191         ERRMSG("Cannot create file!");
03192
03193     /* Write header... */
03194     fprintf(out,
03195             "# $1 = time [s]\n"
03196             "# $2 = altitude [km]\n"
03197             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03198     for (int iq = 0; iq < ctl->nq; iq++)
03199         fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
03200             ctl->qnt_name[iq], ctl->qnt_unit[iq]);
03201     fprintf(out, "\n");
03202
03203     /* Set geolocation and search radius... */
03204     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
03205     rmax2 = SQR(ctl->stat_r);
03206 }
03207
03208 /* Set time interval for output... */
03209 t0 = t - 0.5 * ctl->dt_mod;
03210 t1 = t + 0.5 * ctl->dt_mod;
03211
03212 /* Loop over air parcels... */
03213 for (int ip = 0; ip < atm->np; ip++) {
03214
03215     /* Check time... */
03216     if (atm->time[ip] < t0 || atm->time[ip] > t1)
03217         continue;
03218
03219     /* Check station flag... */
03220     if (ctl->qnt_stat >= 0)
03221         if (atm->q[ctl->qnt_stat][ip])
03222             continue;
03223
03224     /* Get Cartesian coordinates... */
03225     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
03226
03227     /* Check horizontal distance... */
03228     if (DIST2(x0, x1) > rmax2)
03229         continue;
03230
03231     /* Set station flag... */
03232     if (ctl->qnt_stat >= 0)
03233         atm->q[ctl->qnt_stat][ip] = 1;
03234
03235     /* Write data... */
03236     fprintf(out, "%.2f %g %g %g",
03237         atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
03238     for (int iq = 0; iq < ctl->nq; iq++) {
03239         fprintf(out, " ");
03240         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
03241     }
03242     fprintf(out, "\n");
03243 }
03244
03245 /* Close file... */
03246 if (t == ctl->t_stop)
03247     fclose(out);
03248 }

```

Here is the call graph for this function:



5.22 libtrac.h

```

00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00035 #include <ctype.h>
00036 #include <gsl/gsl_math.h>
00037 #include <gsl/gsl_randist.h>
00038 #include <gsl/gsl_rng.h>
00039 #include <gsl/gsl_sort.h>
00040 #include <gsl/gsl_spline.h>
00041 #include <gsl/gsl_statistics.h>
00042 #include <math.h>
00043 #include <netcdf.h>
00044 #include <omp.h>
00045 #include <stdio.h>
00046 #include <stdlib.h>
00047 #include <string.h>
00048 #include <time.h>
00049 #include <sys/time.h>
00050
00051 /* -----
00052   Constants...
00053   ----- */
00054
00056 #define G0 9.80665
00057
00059 #define H0 7.0
00060
00062 #define KB 1.3806504e-23
00063
00065 #define MA 28.9644
00066
00068 #define P0 1013.25
00069
00071 #define RA 287.058
00072
00074 #define RI 8.3144598
00075
00077 #define RE 6367.421
00078
00079 /* -----
00080   Dimensions...
00081   ----- */
00082
00084 #define LEN 5000
00085
00087 #define NP 10000000
00088
00090 #define NQ 12
00091
00093 #define EP 112
00094
00096 #define EX 1201
00097
00099 #define EY 601
00100
00102 #define GX 720
00103
00105 #define GY 360
00106
00108 #define GZ 100
00109
00111 #define NENS 2000
00112
00114 #define NTHREADS 512
00115
00116 /* -----
00117   Macros...
00118   ----- */

```

```

00119
00121 #define ALLOC(ptr, type, n) \
00122     if((ptr=calloc((size_t)(n), sizeof(type)))==NULL) \
00123         ERRMSG("Out of memory!");
00124
00126 #define DEG2DX(dlon, lat) \
00127     ((dlon) * M_PI * RE / 180. * cos((lat) / 180. * M_PI))
00128
00130 #define DEG2DY(dlat) \
00131     ((dlat) * M_PI * RE / 180.)
00132
00134 #define DP2DZ(dp, p) \
00135     (- (dp) * H0 / (p))
00136
00138 #define DX2DEG(dx, lat) \
00139     (((lat) < -89.999 || (lat) > 89.999) ? 0 \
00140      : (dx) * 180. / (M_PI * RE * cos((lat) / 180. * M_PI)))
00141
00143 #define DY2DEG(dy) \
00144     ((dy) * 180. / (M_PI * RE))
00145
00147 #define DZ2DP(dz, p) \
00148     (- (dz) * (p) / H0)
00149
00151 #define DIST(a, b) sqrt(DIST2(a, b))
00152
00154 #define DIST2(a, b) \
00155     ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))
00156
00158 #define DOTP(a, b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
00159
00161 #define ERRMSG(msg) { \
00162     printf("\nError (%s, %s, %d): %s\n\n", \
00163         __FILE__, __func__, __LINE__, msg); \
00164     exit(EXIT_FAILURE); \
00165 }
00166
00168 #define FMOD(x, y) \
00169     ((x) - (int) ((x) / (y)) * (y))
00170
00172 #define FREAD(ptr, type, size, out) { \
00173     if(fread(ptr, sizeof(type), size, out)!=size) \
00174         ERRMSG("Error while reading!"); \
00175 }
00176
00178 #define FWRITE(ptr, type, size, out) { \
00179     if(fwrite(ptr, sizeof(type), size, out)!=size) \
00180         ERRMSG("Error while writing!"); \
00181 }
00182
00184 #define LIN(x0, y0, x1, y1, x) \
00185     ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))
00186
00188 #define NC(cmd) { \
00189     if((cmd)!=NC_NOERR) \
00190         ERRMSG(nc_strerror(cmd)); \
00191 }
00192
00194 #define NORM(a) sqrt(DOTP(a, a))
00195
00197 #define PRINT(format, var) \
00198     printf("Print (%s, %s, %d): %s= "format"\n", \
00199         __FILE__, __func__, __LINE__, #var, var);
00200
00202 #define P(z) (P0*exp(-(z)/H0))
00203
00205 #define SQR(x) ((x)*(x))
00206
00208 #define THETA(p, t) ((t)*pow(1000./(p), 0.286))
00209
00211 #define TOK(line, tok, format, var) { \
00212     if((tok)=strtok((line), " \t")) { \
00213         if(sscanf(tok, format, &(var))!=1) continue; \
00214     } else ERRMSG("Error while reading!"); \
00215 }
00216
00218 #define TVIRT(t, h2o) ((t)*(1.0 + 0.609133 * (h2o) * 18.01528 / MA))
00219
00221 #define WARN(msg) { \
00222     printf("\nWarning (%s, %s, %d): %s\n\n", \
00223         __FILE__, __func__, __LINE__, msg); \
00224 }
00225
00227 #define Z(p) (H0*log(P0/(p)))
00228
00229 /* -----
00230     Timers...

```

```

00231 ----- */
00232
00234 #define START_TIMER(id) timer(#id, id, 1)
00235
00237 #define STOP_TIMER(id) timer(#id, id, 2)
00238
00240 #define PRINT_TIMER(id) timer(#id, id, 3)
00241
00243 #define NTIMER 20
00244
00246 #define TIMER_ZERO 0
00247
00249 #define TIMER_INIT 1
00250
00252 #define TIMER_INPUT 2
00253
00255 #define TIMER_OUTPUT 3
00256
00258 #define TIMER_ADVECT 4
00259
00261 #define TIMER_DECAY 5
00262
00264 #define TIMER_DIFFMESO 6
00265
00267 #define TIMER_DIFFTURB 7
00268
00270 #define TIMER_ISOSURF 8
00271
00273 #define TIMER_METEO 9
00274
00276 #define TIMER_POSITION 10
00277
00279 #define TIMER_SEDI 11
00280
00282 #define TIMER_TOTAL 12
00283
00284 /* -----
00285     Structs...
00286 ----- */
00287
00289 typedef struct {
00290
00292     int nq;
00293
00295     char qnt_name[NQ][LEN];
00296
00298     char qnt_unit[NQ][LEN];
00299
00301     char qnt_format[NQ][LEN];
00302
00304     int qnt_ens;
00305
00307     int qnt_m;
00308
00310     int qnt_rho;
00311
00313     int qnt_r;
00314
00316     int qnt_ps;
00317
00319     int qnt_pt;
00320
00322     int qnt_z;
00323
00325     int qnt_p;
00326
00328     int qnt_t;
00329
00331     int qnt_u;
00332
00334     int qnt_v;
00335
00337     int qnt_w;
00338
00340     int qnt_h2o;
00341
00343     int qnt_o3;
00344
00346     int qnt_theta;
00347
00349     int qnt_vh;
00350
00352     int qnt_vz;
00353
00355     int qnt_pv;
00356
00358     int qnt_tice;

```

```
00359
00361     int qnt_tsts;
00362
00364     int qnt_tnat;
00365
00367     int qnt_stat;
00368
00370     int direction;
00371
00373     double t_start;
00374
00376     double t_stop;
00377
00379     double dt_mod;
00380
00382     double dt_met;
00383
00385     int met_dx;
00386
00388     int met_dy;
00389
00391     int met_dp;
00392
00394     int met_sx;
00395
00397     int met_sy;
00398
00400     int met_sp;
00401
00403     int met_np;
00404
00406     double met_p[EP];
00407
00410     int met_tropo;
00411
00413     char met_geopot[LEN];
00414
00416     double met_dt_out;
00417
00419     char met_stage[LEN];
00420
00423     int isosurf;
00424
00426     char balloon[LEN];
00427
00429     double turb_dx_trop;
00430
00432     double turb_dx_strat;
00433
00435     double turb_dz_trop;
00436
00438     double turb_dz_strat;
00439
00441     double turb_mesox;
00442
00444     double turb_mesoz;
00445
00447     double molmass;
00448
00450     double tdec_trop;
00451
00453     double tdec_strat;
00454
00456     double psc_h2o;
00457
00459     double psc_hno3;
00460
00462     char atm_basename[LEN];
00463
00465     char atm_gpfile[LEN];
00466
00468     double atm_dt_out;
00469
00471     int atm_filter;
00472
00474     int atm_stride;
00475
00477     int atm_type;
00478
00480     char csi_basename[LEN];
00481
00483     double csi_dt_out;
00484
00486     char csi_obsfile[LEN];
00487
00489     double csi_obsmin;
00490
```

```
00492 double csi_modmin;
00493
00495 int csi_nz;
00496
00498 double csi_z0;
00499
00501 double csi_z1;
00502
00504 int csi_nx;
00505
00507 double csi_lon0;
00508
00510 double csi_lon1;
00511
00513 int csi_ny;
00514
00516 double csi_lat0;
00517
00519 double csi_lat1;
00520
00522 char grid_basename[LEN];
00523
00525 char grid_gpfile[LEN];
00526
00528 double grid_dt_out;
00529
00531 int grid_sparse;
00532
00534 int grid_nz;
00535
00537 double grid_z0;
00538
00540 double grid_z1;
00541
00543 int grid_nx;
00544
00546 double grid_lon0;
00547
00549 double grid_lon1;
00550
00552 int grid_ny;
00553
00555 double grid_lat0;
00556
00558 double grid_lat1;
00559
00561 char prof_basename[LEN];
00562
00564 char prof_obsfile[LEN];
00565
00567 int prof_nz;
00568
00570 double prof_z0;
00571
00573 double prof_z1;
00574
00576 int prof_nx;
00577
00579 double prof_lon0;
00580
00582 double prof_lon1;
00583
00585 int prof_ny;
00586
00588 double prof_lat0;
00589
00591 double prof_lat1;
00592
00594 char ens_basename[LEN];
00595
00597 char stat_basename[LEN];
00598
00600 double stat_lon;
00601
00603 double stat_lat;
00604
00606 double stat_r;
00607
00608 } ctl_t;
00609
00611 typedef struct {
00612
00614 int np;
00615
00617 double time[NP];
00618
00620 double p[NP];
```

```

00621
00623     double lon[NP];
00624
00626     double lat[NP];
00627
00629     double q[NQ][NP];
00630
00631 } atm_t;
00632
00634 typedef struct {
00635
00637     float up[NP];
00638
00640     float vp[NP];
00641
00643     float wp[NP];
00644
00646     double iso_var[NP];
00647
00649     double iso_ps[NP];
00650
00652     double iso_ts[NP];
00653
00655     int iso_n;
00656
00658     double tsig[EX][EY][EP];
00659
00661     float usig[EX][EY][EP];
00662
00664     float vsig[EX][EY][EP];
00665
00667     float wsig[EX][EY][EP];
00668
00669 } cache_t;
00670
00672 typedef struct {
00673
00675     double time;
00676
00678     int nx;
00679
00681     int ny;
00682
00684     int np;
00685
00687     double lon[EX];
00688
00690     double lat[EY];
00691
00693     double p[EP];
00694
00696     double ps[EX][EY];
00697
00699     double pt[EX][EY];
00700
00702     float z[EX][EY][EP];
00703
00705     float t[EX][EY][EP];
00706
00708     float u[EX][EY][EP];
00709
00711     float v[EX][EY][EP];
00712
00714     float w[EX][EY][EP];
00715
00717     float pv[EX][EY][EP];
00718
00720     float h2o[EX][EY][EP];
00721
00723     float o3[EX][EY][EP];
00724
00726     float pl[EX][EY][EP];
00727
00728 } met_t;
00729
00730 /* -----
00731     Functions...
00732     ----- */
00733
00735 void cart2geo(
00736     double *x,
00737     double *z,
00738     double *lon,
00739     double *lat);
00740
00742 #ifdef _OPENACC
00743 #pragma acc routine (clim_hno3)

```

```
00744 #endif
00745 double clim_hno3(
00746     double t,
00747     double lat,
00748     double p);
00749
00751 #ifdef _OPENACC
00752 #pragma acc routine (clim_tropo)
00753 #endif
00754 double clim_tropo(
00755     double t,
00756     double lat);
00757
00759 void day2doy(
00760     int year,
00761     int mon,
00762     int day,
00763     int *doy);
00764
00766 void doy2day(
00767     int year,
00768     int doy,
00769     int *mon,
00770     int *day);
00771
00773 void geo2cart(
00774     double z,
00775     double lon,
00776     double lat,
00777     double *x);
00778
00780 void get_met(
00781     ctl_t * ctl,
00782     char *metbase,
00783     double t,
00784     met_t ** met0,
00785     met_t ** met1);
00786
00788 void get_met_help(
00789     double t,
00790     int direct,
00791     char *metbase,
00792     double dt_met,
00793     char *filename);
00794
00796 void get_met_replace(
00797     char *orig,
00798     char *search,
00799     char *repl);
00800
00802 #ifdef _OPENACC
00803 #pragma acc routine (intpol_met_2d)
00804 #endif
00805 double intpol_met_2d(
00806     double array[EX][EY],
00807     int ix,
00808     int iy,
00809     double wx,
00810     double wy);
00811
00813 #ifdef _OPENACC
00814 #pragma acc routine (intpol_met_3d)
00815 #endif
00816 double intpol_met_3d(
00817     float array[EX][EY][EP],
00818     int ip,
00819     int ix,
00820     int iy,
00821     double wp,
00822     double wx,
00823     double wy);
00824
00826 #ifdef _OPENACC
00827 #pragma acc routine (intpol_met_space)
00828 #endif
00829 void intpol_met_space(
00830     met_t * met,
00831     double p,
00832     double lon,
00833     double lat,
00834     double *ps,
00835     double *pt,
00836     double *z,
00837     double *t,
00838     double *u,
00839     double *v,
00840     double *w,
```



```
00841 double *pv,
00842 double *h2o,
00843 double *o3);
00844
00846 #ifdef _OPENACC
00847 #pragma acc routine (intpol_met_time)
00848 #endif
00849 void intpol_met_time(
00850     met_t * met0,
00851     met_t * met1,
00852     double ts,
00853     double p,
00854     double lon,
00855     double lat,
00856     double *ps,
00857     double *pt,
00858     double *z,
00859     double *t,
00860     double *u,
00861     double *v,
00862     double *w,
00863     double *pv,
00864     double *h2o,
00865     double *o3);
00866
00868 void jsec2time(
00869     double jsec,
00870     int *year,
00871     int *mon,
00872     int *day,
00873     int *hour,
00874     int *min,
00875     int *sec,
00876     double *remain);
00877
00879 #ifdef _OPENACC
00880 #pragma acc routine (locate_irr)
00881 #endif
00882 int locate_irr(
00883     double **xx,
00884     int n,
00885     double x);
00886
00888 #ifdef _OPENACC
00889 #pragma acc routine (locate_reg)
00890 #endif
00891 int locate_reg(
00892     double **xx,
00893     int n,
00894     double x);
00895
00897 int read_atm(
00898     const char *filename,
00899     ctl_t * ctl,
00900     atm_t * atm);
00901
00903 void read_ctl(
00904     const char *filename,
00905     int argc,
00906     char *argv[],
00907     ctl_t * ctl);
00908
00910 int read_met(
00911     ctl_t * ctl,
00912     char *filename,
00913     met_t * met);
00914
00916 void read_met_extrapolate(
00917     met_t * met);
00918
00920 void read_met_geopot(
00921     ctl_t * ctl,
00922     met_t * met);
00923
00925 void read_met_help(
00926     int ncid,
00927     char *varname,
00928     char *varname2,
00929     met_t * met,
00930     float dest[EX][EY][EP],
00931     float scl);
00932
00934 void read_met_ml2pl(
00935     ctl_t * ctl,
00936     met_t * met,
00937     float var[EX][EY][EP]);
00938
```

```
00940 void read_met_periodic(
00941     met_t * met);
00942
00944 void read_met_pv(
00945     met_t * met);
00946
00948 void read_met_sample(
00949     ctl_t * ctl,
00950     met_t * met);
00951
00953 void read_met_tropo(
00954     ctl_t * ctl,
00955     met_t * met);
00956
00958 double scan_ctl(
00959     const char *filename,
00960     int argc,
00961     char *argv[],
00962     const char *varname,
00963     int arridx,
00964     const char *defvalue,
00965     char *value);
00966
00968 void spline(
00969     double *x,
00970     double *y,
00971     int n,
00972     double *x2,
00973     double *y2,
00974     int n2);
00975
00977 #ifdef _OPENACC
00978 #pragma acc routine (stddev)
00979 #endif
00980 double stddev(
00981     double *data,
00982     int n);
00983
00985 void time2jsec(
00986     int year,
00987     int mon,
00988     int day,
00989     int hour,
00990     int min,
00991     int sec,
00992     double remain,
00993     double *jsec);
00994
00996 void timer(
00997     const char *name,
00998     int id,
00999     int mode);
01000
01002 void write_atm(
01003     const char *filename,
01004     ctl_t * ctl,
01005     atm_t * atm,
01006     double t);
01007
01009 void write_csi(
01010     const char *filename,
01011     ctl_t * ctl,
01012     atm_t * atm,
01013     double t);
01014
01016 void write_ens(
01017     const char *filename,
01018     ctl_t * ctl,
01019     atm_t * atm,
01020     double t);
01021
01023 void write_grid(
01024     const char *filename,
01025     ctl_t * ctl,
01026     met_t * met0,
01027     met_t * met1,
01028     atm_t * atm,
01029     double t);
01030
01032 void write_prof(
01033     const char *filename,
01034     ctl_t * ctl,
01035     met_t * met0,
01036     met_t * met1,
01037     atm_t * atm,
01038     double t);
01039
```

```

01041 void write_station(
01042     const char *filename,
01043     ctl_t *ctl,
01044     atm_t *atm,
01045     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], o3, o3m[NX][NY], z, zm[NX][NY], pv, pvm[NX][NY], zt,
00054         ztm[NX][NY], tt, ttm[NX][NY], lon, lon0, lon1, lons[NX], dlon, lat, lat0,
00055         lat1, lats[NY], dlat;
00056
00057     static int i, ix, iy, np[NX][NY], nx, ny;
00058
00059     /* Allocate... */
00060     ALLOC(met, met_t, 1);
00061
00062     /* Check arguments... */
00063     if (argc < 4)
00064         ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00065
00066     /* Read control parameters... */
00067     read_ctl(argv[1], argc, argv, &ctl);
00068     p0 = P(scan_ctl(argv[1], argc, argv, "MAP_Z0", -1, "", NULL));
00069     lon0 = scan_ctl(argv[1], argc, argv, "MAP_LON0", -1, "-180", NULL);
00070     lon1 = scan_ctl(argv[1], argc, argv, "MAP_LON1", -1, "180", NULL);
00071     dlon = scan_ctl(argv[1], argc, argv, "MAP_DLON", -1, "-999", NULL);
00072     lat0 = scan_ctl(argv[1], argc, argv, "MAP_LAT0", -1, "-90", NULL);
00073     lat1 = scan_ctl(argv[1], argc, argv, "MAP_LAT1", -1, "90", NULL);
00074     dlat = scan_ctl(argv[1], argc, argv, "MAP_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 horizontal grid... */
00084 if (dlon <= 0)
00085     dlon = fabs(met->lon[1] - met->lon[0]);
00086 if (dlat <= 0)
00087     dlat = fabs(met->lat[1] - met->lat[0]);
00088 if (lon0 < -360 && lon1 > 360) {
00089     lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00090     lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00091 }
00092 nx = ny = 0;
00093 for (lon = lon0; lon <= lon1; lon += dlon) {
00094     lons[nx] = lon;
00095     if ((++nx) > NX)
00096         ERRMSG("Too many longitudes!");
00097 }
00098 if (lat0 < -90 && lat1 > 90) {
00099     lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00100     lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00101 }
00102 for (lat = lat0; lat <= lat1; lat += dlat) {
00103     lats[ny] = lat;
00104     if ((++ny) > NY)
00105         ERRMSG("Too many latitudes!");
00106 }
00107
00108 /* Average... */
00109 for (ix = 0; ix < nx; ix++)
00110     for (iy = 0; iy < ny; iy++) {
00111         intpol_met_space(met, p0, lons[ix], lats[iy], &ps, &pt,
00112                         &z, &t, &u, &v, &w, &pv, &h2o, &o3);
00113         intpol_met_space(met, pt, lons[ix], lats[iy], NULL, NULL,
00114                         &zt, &tt, NULL, NULL, NULL, NULL, NULL);
00115         timem[ix][iy] += met->time;
00116         zm[ix][iy] += z;
00117         tm[ix][iy] += t;
00118         um[ix][iy] += u;
00119         vm[ix][iy] += v;
00120         wm[ix][iy] += w;
00121         pvm[ix][iy] += pv;
00122         h2om[ix][iy] += h2o;
00123         o3m[ix][iy] += o3;
00124         psm[ix][iy] += ps;
00125         ptm[ix][iy] += pt;
00126         ztm[ix][iy] += zt;
00127         ttm[ix][iy] += tt;
00128         np[ix][iy]++;
00129     }
00130 }
00131
00132 /* Create output file... */
00133 printf("Write meteorological data file: %s\n", argv[2]);
00134 if (!(out = fopen(argv[2], "w")))
00135     ERRMSG("Cannot create file!");
00136
00137 /* Write header... */
00138 fprintf(out,
00139         "# $1 = time [s]\n"
00140         "# $2 = altitude [km]\n"
00141         "# $3 = longitude [deg]\n"
00142         "# $4 = latitude [deg]\n"
00143         "# $5 = pressure [hPa]\n"
00144         "# $6 = temperature [K]\n"
00145         "# $7 = zonal wind [m/s]\n"
00146         "# $8 = meridional wind [m/s]\n" "# $9 = vertical wind [hPa/s]\n");
00147 fprintf(out,
00148         "# $10 = H2O volume mixing ratio [1]\n"
00149         "# $11 = O3 volume mixing ratio [1]\n"
00150         "# $12 = geopotential height [km]\n"
00151         "# $13 = potential vorticity [PVU]\n"
00152         "# $14 = surface pressure [hPa]\n"
00153         "# $15 = tropopause pressure [hPa]\n"
00154         "# $16 = tropopause geopotential height [km]\n"
00155         "# $17 = tropopause temperature [K]\n");
00156
00157 /* Write data... */
00158 for (iy = 0; iy < ny; iy++) {
00159     fprintf(out, "\n");
00160     for (ix = 0; ix < nx; ix++)
00161         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g\n",
00162                 timem[ix][iy] / np[ix][iy], Z(p0), lons[ix], lats[iy], p0,
00163                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00164                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00165                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00166                 zm[ix][iy] / np[ix][iy], pvm[ix][iy] / np[ix][iy],
00167                 psm[ix][iy] / np[ix][iy], ptm[ix][iy] / np[ix][iy],
00168                 ztm[ix][iy] / np[ix][iy], ttm[ix][iy] / np[ix][iy]);

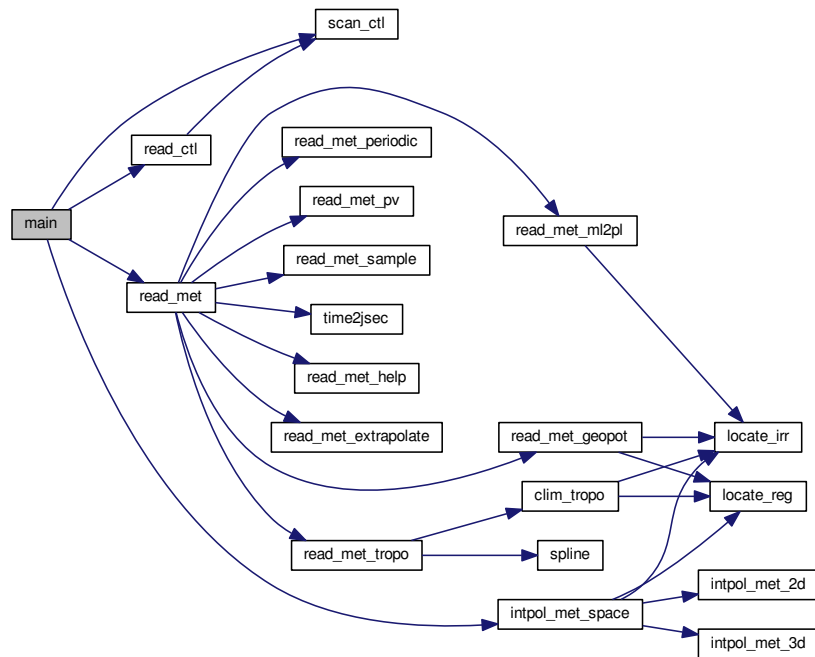
```

```

00169     }
00170
00171     /* Close file... */
00172     fclose(out);
00173
00174     /* Free... */
00175     free(met);
00176
00177     return EXIT_SUCCESS;
00178 }

```

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], o3, o3m[NX][NY], z, zm[NX][NY], pv, pvm[NX][NY], zt,
00054         ztm[NX][NY], tt, ttm[NX][NY], lon, lon0, lon1, lons[NX], dlon, lat, lat0,
00055         lat1, lats[NY], dlat;
00056
00057     static int i, ix, iy, np[NX][NY], nx, ny;
00058
00059     /* Allocate... */
00060     ALLOC(met, met_t, 1);
00061
00062     /* Check arguments... */
00063     if (argc < 4)
00064         ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00065
00066     /* Read control parameters... */
00067     read_ctl(argv[1], argc, argv, &ctl);
00068     p0 = P(scan_ctl(argv[1], argc, argv, "MAP_Z0", -1, "", NULL));
00069     lon0 = scan_ctl(argv[1], argc, argv, "MAP_LON0", -1, "-180", NULL);
00070     lon1 = scan_ctl(argv[1], argc, argv, "MAP_LON1", -1, "180", NULL);
00071     dlon = scan_ctl(argv[1], argc, argv, "MAP_DLON", -1, "-999", NULL);
00072     lat0 = scan_ctl(argv[1], argc, argv, "MAP_LAT0", -1, "-90", NULL);
00073     lat1 = scan_ctl(argv[1], argc, argv, "MAP_LAT1", -1, "90", NULL);
00074     dlat = scan_ctl(argv[1], argc, argv, "MAP_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 horizontal grid... */
00084         if (dlon <= 0)
00085             dlon = fabs(met->lon[1] - met->lon[0]);
00086         if (dlat <= 0)
00087             dlat = fabs(met->lat[1] - met->lat[0]);
00088         if (lon0 < -360 && lon1 > 360) {
00089             lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00090             lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00091         }
00092         nx = ny = 0;
00093         for (lon = lon0; lon <= lon1; lon += dlon) {
00094             lons[nx] = lon;
00095             if ((++nx) > NX)
00096                 ERRMSG("Too many longitudes!");
00097         }
00098         if (lat0 < -90 && lat1 > 90) {
00099             lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00100             lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00101         }
00102         for (lat = lat0; lat <= lat1; lat += dlat) {
00103             lats[ny] = lat;
00104             if ((++ny) > NY)
00105                 ERRMSG("Too many latitudes!");
00106         }
00107
00108         /* Average... */
00109         for (ix = 0; ix < nx; ix++)
00110             for (iy = 0; iy < ny; iy++) {
00111                 intpol_met_space(met, p0, lons[ix], lats[iy], &ps, &pt,
00112                     &z, &t, &u, &v, &w, &pv, &h2o, &o3);
00113                 intpol_met_space(met, pt, lons[ix], lats[iy], NULL, NULL,
00114                     &zt, &tt, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
00115                 timem[ix][iy] += met->time;
00116                 zm[ix][iy] += z;
00117                 tm[ix][iy] += t;
00118                 um[ix][iy] += u;
00119                 vm[ix][iy] += v;
00120                 wm[ix][iy] += w;

```

```

00121         pvm[ix][iy] += pv;
00122         h2om[ix][iy] += h2o;
00123         o3m[ix][iy] += o3;
00124         psm[ix][iy] += ps;
00125         ptm[ix][iy] += pt;
00126         ztm[ix][iy] += zt;
00127         ttm[ix][iy] += tt;
00128         np[ix][iy]++;
00129     }
00130 }
00131
00132 /* Create output file... */
00133 printf("Write meteorological data file: %s\n", argv[2]);
00134 if (! (out = fopen(argv[2], "w")))
00135     ERRMSG("Cannot create file!");
00136
00137 /* Write header... */
00138 fprintf(out,
00139         "# $1 = time [s]\n"
00140         "# $2 = altitude [km]\n"
00141         "# $3 = longitude [deg]\n"
00142         "# $4 = latitude [deg]\n"
00143         "# $5 = pressure [hPa]\n"
00144         "# $6 = temperature [K]\n"
00145         "# $7 = zonal wind [m/s]\n"
00146         "# $8 = meridional wind [m/s]\n" "# $9 = vertical wind [hPa/s]\n");
00147 fprintf(out,
00148         "# $10 = H2O volume mixing ratio [1]\n"
00149         "# $11 = O3 volume mixing ratio [1]\n"
00150         "# $12 = geopotential height [km]\n"
00151         "# $13 = potential vorticity [PVU]\n"
00152         "# $14 = surface pressure [hPa]\n"
00153         "# $15 = tropopause pressure [hPa]\n"
00154         "# $16 = tropopause geopotential height [km]\n"
00155         "# $17 = tropopause temperature [K]\n");
00156
00157 /* Write data... */
00158 for (iy = 0; iy < ny; iy++) {
00159     fprintf(out, "\n");
00160     for (ix = 0; ix < nx; ix++)
00161         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g\n",
00162                 timem[ix][iy] / np[ix][iy], Z(p0), lons[ix], lats[iy], p0,
00163                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00164                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00165                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00166                 zm[ix][iy] / np[ix][iy], pvm[ix][iy] / np[ix][iy],
00167                 psm[ix][iy] / np[ix][iy], ptm[ix][iy] / np[ix][iy],
00168                 ztm[ix][iy] / np[ix][iy], ttm[ix][iy] / np[ix][iy]);
00169 }
00170
00171 /* Close file... */
00172 fclose(out);
00173
00174 /* Free... */
00175 free(met);
00176
00177 return EXIT_SUCCESS;
00178 }

```

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], o3, o3m[NZ], ps, psm[NZ], pt, ptm[NZ], tt, ttm[NZ],
00051         zm[NZ], zt, ztm[NZ], pv, pvm[NZ], plev[NZ];
00052
00053     static int i, iz, np[NZ], npt[NZ], nz;
00054
00055     /* Allocate... */
00056     ALLOC(met, met_t, 1);
00057
00058     /* Check arguments... */
00059     if (argc < 4)
00060         ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00061
00062     /* Read control parameters... */
00063     read_ctl(argv[1], argc, argv, &ctl);
00064     z0 = scan_ctl(argv[1], argc, argv, "PROF_Z0", -1, "-999", NULL);
00065     z1 = scan_ctl(argv[1], argc, argv, "PROF_Z1", -1, "-999", NULL);
00066     dz = scan_ctl(argv[1], argc, argv, "PROF_DZ", -1, "-999", NULL);
00067     lon0 = scan_ctl(argv[1], argc, argv, "PROF_LON0", -1, "", NULL);
00068     lon1 = scan_ctl(argv[1], argc, argv, "PROF_LON1", -1, "", NULL);
00069     dlon = scan_ctl(argv[1], argc, argv, "PROF_DLON", -1, "-999", NULL);
00070     lat0 = scan_ctl(argv[1], argc, argv, "PROF_LAT0", -1, "", NULL);
00071     lat1 = scan_ctl(argv[1], argc, argv, "PROF_LAT1", -1, "", NULL);
00072     dlat = scan_ctl(argv[1], argc, argv, "PROF_DLAT", -1, "-999", NULL);
00073
00074     /* Loop over input files... */
00075     for (i = 3; i < argc; i++) {
00076
00077         /* Read meteorological data... */
00078         if (!read_met(&ctl, argv[i], met))
00079             continue;
00080
00081         /* Set vertical grid... */
00082         if (z0 < 0)
00083             z0 = Z(met->p[0]);
00084         if (z1 < 0)
00085             z1 = Z(met->p[met->np - 1]);
00086         nz = 0;
00087         if (dz < 0) {
00088             for (iz = 0; iz < met->np; iz++)
00089                 if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00090                     plev[nz] = met->p[iz];
00091                     if (++nz > NZ)
00092                         ERRMSG("Too many pressure levels!");
00093                 }
00094             } else
00095                 for (z = z0; z <= z1; z += dz) {
00096                     plev[nz] = P(z);
00097                     if (++nz > NZ)
00098                         ERRMSG("Too many pressure levels!");
00099                 }
00100
00101         /* Set horizontal grid... */
00102         if (dlon <= 0)
00103             dlon = fabs(met->lon[1] - met->lon[0]);
00104         if (dlat <= 0)
00105             dlat = fabs(met->lat[1] - met->lat[0]);
00106
00107         /* Average... */
00108         for (iz = 0; iz < nz; iz++)
00109             for (lon = lon0; lon <= lon1; lon += dlon)
00110                 for (lat = lat0; lat <= lat1; lat += dlat) {
00111                     intpol_met_space(met, plev[iz], lon, lat, &ps, &pt, &z,
00112                                     &t, &u, &v, &w, &pv, &h2o, &o3);
00113                     intpol_met_space(met, pt, lon, lat, NULL, NULL, &zt,
00114                                     &tt, NULL, NULL, NULL, NULL, NULL, NULL);
00115                     if (gsl_finite(t) && gsl_finite(u)

```

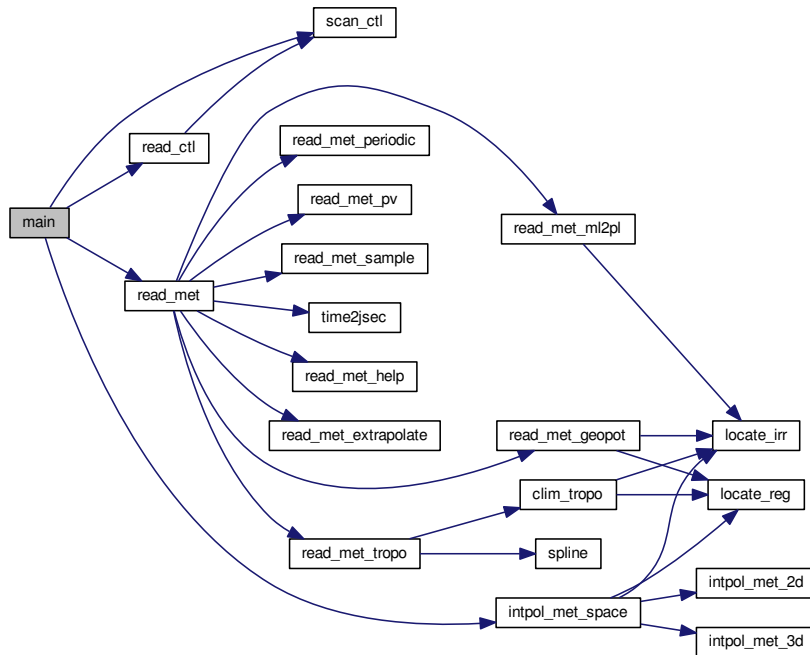


```

00116         && gsl_finite(v) && gsl_finite(w)) {
00117             timem[iz] += met->time;
00118             lonm[iz] += lon;
00119             latm[iz] += lat;
00120             zm[iz] += z;
00121             tm[iz] += t;
00122             um[iz] += u;
00123             vm[iz] += v;
00124             wm[iz] += w;
00125             pvm[iz] += pv;
00126             h2om[iz] += h2o;
00127             o3m[iz] += o3;
00128             psm[iz] += ps;
00129             if (gsl_finite(pt)) {
00130                 ptm[iz] += pt;
00131                 ztm[iz] += zt;
00132                 ttm[iz] += tt;
00133                 npt[iz]++;
00134             }
00135             np[iz]++;
00136         }
00137     }
00138 }
00139
00140 /* Create output file... */
00141 printf("Write meteorological data file: %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"
00150         "# $4 = latitude [deg]\n"
00151         "# $5 = pressure [hPa]\n"
00152         "# $6 = temperature [K]\n"
00153         "# $7 = zonal wind [m/s]\n"
00154         "# $8 = meridional wind [m/s]\n" "# $9 = vertical wind [hPa/s]\n");
00155 fprintf(out,
00156         "# $10 = H2O volume mixing ratio [1]\n"
00157         "# $11 = O3 volume mixing ratio [1]\n"
00158         "# $12 = geopotential height [km]\n"
00159         "# $13 = potential vorticity [PVU]\n"
00160         "# $14 = surface pressure [hPa]\n"
00161         "# $15 = tropopause pressure [hPa]\n"
00162         "# $16 = tropopause geopotential height [km]\n"
00163         "# $17 = tropopause temperature [K]\n\n");
00164
00165 /* Write data... */
00166 for (iz = 0; iz < nz; iz++)
00167     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00168             timem[iz] / np[iz], Z(plev[iz]), lonm[iz] / np[iz],
00169             latm[iz] / np[iz], plev[iz], tm[iz] / np[iz], um[iz] / np[iz],
00170             vm[iz] / np[iz], wm[iz] / np[iz], h2om[iz] / np[iz],
00171             o3m[iz] / np[iz], zm[iz] / np[iz], pvm[iz] / np[iz],
00172             psm[iz] / np[iz], ptm[iz] / npt[iz], ztm[iz] / npt[iz],
00173             ttm[iz] / npt[iz]);
00174
00175 /* Close file... */
00176 fclose(out);
00177
00178 /* Free... */
00179 free(met);
00180
00181 return EXIT_SUCCESS;
00182 }

```

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], o3, o3m[NZ], ps, psm[NZ], pt, ptm[NZ], tt, ttm[NZ],
00051 zm[NZ], zt, ztm[NZ], pv, pvm[NZ], plev[NZ];
00052
00053 static int i, iz, np[NZ], npt[NZ], nz;
00054
00055 /* Allocate... */
00056 ALLOC(met, met_t, 1);
00057
00058 /* Check arguments... */
00059 if (argc < 4)
00060     ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00061
00062 /* Read control parameters... */
00063 read_ctl(argv[1], argc, argv, &ctl);
00064 z0 = scan_ctl(argv[1], argc, argv, "PROF_Z0", -1, "-999", NULL);
00065 z1 = scan_ctl(argv[1], argc, argv, "PROF_Z1", -1, "-999", NULL);
00066 dz = scan_ctl(argv[1], argc, argv, "PROF_DZ", -1, "-999", NULL);
00067 lon0 = scan_ctl(argv[1], argc, argv, "PROF_LON0", -1, "", NULL);
00068 lon1 = scan_ctl(argv[1], argc, argv, "PROF_LON1", -1, "", NULL);
00069 dlon = scan_ctl(argv[1], argc, argv, "PROF_DLON", -1, "-999", NULL);
00070 lat0 = scan_ctl(argv[1], argc, argv, "PROF_LAT0", -1, "", NULL);
00071 lat1 = scan_ctl(argv[1], argc, argv, "PROF_LAT1", -1, "", NULL);
00072 dlat = scan_ctl(argv[1], argc, argv, "PROF_DLAT", -1, "-999", NULL);
00073
00074 /* Loop over input files... */
00075 for (i = 3; i < argc; i++) {
00076
00077     /* Read meteorological data... */
00078     if (!read_met(&ctl, argv[i], met))
00079         continue;
00080
00081     /* Set vertical grid... */
00082     if (z0 < 0)
00083         z0 = Z(met->p[0]);
00084     if (z1 < 0)
00085         z1 = Z(met->p[met->np - 1]);
00086     nz = 0;
00087     if (dz < 0) {
00088         for (iz = 0; iz < met->np; iz++)
00089             if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00090                 plev[nz] = met->p[iz];
00091                 if (++nz > NZ)
00092                     ERRMSG("Too many pressure levels!");
00093             }
00094     } else
00095         for (z = z0; z <= z1; z += dz) {
00096             plev[nz] = P(z);
00097             if (++nz > NZ)
00098                 ERRMSG("Too many pressure levels!");
00099         }
00100
00101     /* Set horizontal grid... */
00102     if (dlon <= 0)
00103         dlon = fabs(met->lon[1] - met->lon[0]);
00104     if (dlat <= 0)
00105         dlat = fabs(met->lat[1] - met->lat[0]);
00106
00107     /* Average... */
00108     for (iz = 0; iz < nz; iz++)
00109         for (lon = lon0; lon <= lon1; lon += dlon)
00110             for (lat = lat0; lat <= lat1; lat += dlat) {
00111                 intpol_met_space(met, plev[iz], lon, lat, &ps, &pt, &z,
00112                                &t, &u, &v, &w, &pv, &h2o, &o3);
00113                 intpol_met_space(met, pt, lon, lat, NULL, NULL, &zt,
00114                                &tt, NULL, NULL, NULL, NULL, NULL, NULL);
00115                 if (gsl_finite(t) && gsl_finite(u)
00116                     && gsl_finite(v) && gsl_finite(w)) {
00117                     timem[iz] += met->time;
00118                     lonm[iz] += lon;
00119                     latm[iz] += lat;
00120                     zm[iz] += z;
00121                     tm[iz] += t;
00122                     um[iz] += u;
00123                     vm[iz] += v;
00124                     wm[iz] += w;
00125                     pvm[iz] += pv;
00126                     h2om[iz] += h2o;
00127                     o3m[iz] += o3;
00128                     psm[iz] += ps;
00129                     if (gsl_finite(pt)) {
00130                         ptm[iz] += pt;
00131                         ztm[iz] += zt;
00132                         ttm[iz] += tt;

```

```

00133         npt[iz]++;
00134     }
00135     np[iz]++;
00136 }
00137 }
00138 }
00139
00140 /* Create output file... */
00141 printf("Write meteorological data file: %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"
00150     "# $4 = latitude [deg]\n"
00151     "# $5 = pressure [hPa]\n"
00152     "# $6 = temperature [K]\n"
00153     "# $7 = zonal wind [m/s]\n"
00154     "# $8 = meridional wind [m/s]\n" "# $9 = vertical wind [hPa/s]\n");
00155 fprintf(out,
00156     "# $10 = H2O volume mixing ratio [1]\n"
00157     "# $11 = O3 volume mixing ratio [1]\n"
00158     "# $12 = geopotential height [km]\n"
00159     "# $13 = potential vorticity [PVU]\n"
00160     "# $14 = surface pressure [hPa]\n"
00161     "# $15 = tropopause pressure [hPa]\n"
00162     "# $16 = tropopause geopotential height [km]\n"
00163     "# $17 = tropopause temperature [K]\n\n");
00164
00165 /* Write data... */
00166 for (iz = 0; iz < nz; iz++)
00167     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00168         timem[iz] / np[iz], Z(plev[iz]), lonm[iz] / np[iz],
00169         latm[iz] / np[iz], plev[iz], tm[iz] / np[iz], um[iz] / np[iz],
00170         vm[iz] / np[iz], wm[iz] / np[iz], h2om[iz] / np[iz],
00171         o3m[iz] / np[iz], zm[iz] / np[iz], pvm[iz] / np[iz],
00172         psm[iz] / np[iz], ptm[iz] / npt[iz], ztm[iz] / npt[iz],
00173         ttm[iz] / npt[iz]);
00174
00175 /* Close file... */
00176 fclose(out);
00177
00178 /* Free... */
00179 free(met);
00180
00181 return EXIT_SUCCESS;
00182 }

```

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, o3, p0, p1, pref, ps, pt, pv, t, tt, u, v, w, z, zm, zref, zt;
00044
00045     int geopot, ip, it;
00046
00047     /* Check arguments... */
00048     if (argc < 4)
00049         ERRMSG("Give parameters: <ctl> <sample.tab> <metbase> <atm_in>");
00050
00051     /* Allocate... */
00052     ALLOC(atm, atm_t, 1);
00053     ALLOC(met0, met_t, 1);
00054     ALLOC(met1, met_t, 1);
00055
00056     /* Read control parameters... */
00057     read_ctl(argv[1], argc, argv, &ctl);
00058     geopot =
00059         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GEOPOT", -1, "0", NULL);
00060
00061     /* Read atmospheric data... */
00062     if (!read_atm(argv[4], &ctl, atm))
00063         ERRMSG("Cannot open file!");
00064
00065     /* Create output file... */
00066     printf("Write meteorological data file: %s\n", argv[2]);
00067     if (!(out = fopen(argv[2], "w")))
00068         ERRMSG("Cannot create file!");
00069
00070     /* Write header... */
00071     fprintf(out,
00072         "# $1 = time [s]\n"
00073         "# $2 = altitude [km]\n"
00074         "# $3 = longitude [deg]\n"
00075         "# $4 = latitude [deg]\n"
00076         "# $5 = pressure [hPa]\n"
00077         "# $6 = temperature [K]\n"
00078         "# $7 = zonal wind [m/s]\n"
00079         "# $8 = meridional wind [m/s]\n"
00080         "# $9 = vertical wind [hPa/s]\n");
00081     fprintf(out,
00082         "# $10 = H2O volume mixing ratio [1]\n"
00083         "# $11 = O3 volume mixing ratio [1]\n"
00084         "# $12 = geopotential height [km]\n"
00085         "# $13 = potential vorticity [PVU]\n"
00086         "# $14 = surface pressure [hPa]\n"
00087         "# $15 = tropopause pressure [hPa]\n"
00088         "# $16 = tropopause geopotential height [km]\n"
00089         "# $17 = tropopause temperature [K]\n\n");
00090
00091     /* Loop over air parcels... */
00092     for (ip = 0; ip < atm->np; ip++) {
00093
00094         /* Get meteorological data... */
00095         get_met(&ctl, argv[3], atm->time[ip], &met0, &met1);
00096
00097         /* Set reference pressure for interpolation... */
00098         pref = atm->p[ip];
00099         if (geopot) {
00100             zref = Z(pref);
00101             p0 = met0->p[0];
00102             p1 = met0->p[met0->np - 1];
00103             for (it = 0; it < 24; it++) {
00104                 pref = 0.5 * (p0 + p1);
00105                 intpol_met_time(met0, met1, atm->time[ip], pref, atm->
lon[ip],
00106                                     atm->lat[ip], NULL, NULL, &zm, NULL, NULL, NULL, NULL,
00107                                     NULL, NULL, NULL);

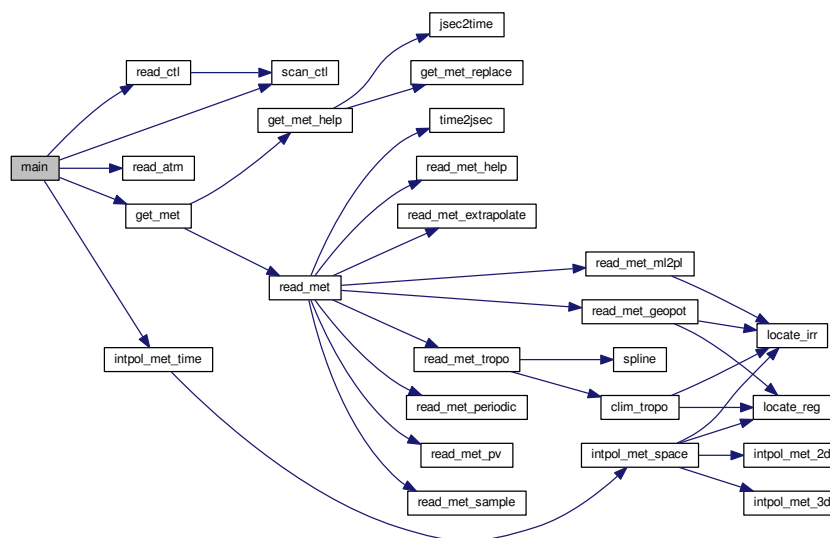
```

```

00108         if (zref > zm || !gsl_finite(zm))
00109             p0 = pref;
00110         else
00111             p1 = pref;
00112     }
00113     pref = 0.5 * (p0 + p1);
00114 }
00115
00116 /* Interpolate meteorological data... */
00117 intpol_met_time(met0, met1, atm->time[ip], pref, atm->lon[ip],
00118                atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &pv, &h2o,
00119                &o3);
00120 intpol_met_time(met0, met1, atm->time[ip], pt, atm->lon[ip], atm->
00121 lat[ip],
00122                NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL, NULL);
00123
00124 /* Write data... */
00125 fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00126         atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
00127         atm->p[ip], t, u, v, w, h2o, o3, z, pv, ps, pt, zt, tt);
00128 }
00129 /* Close file... */
00130 fclose(out);
00131
00132 /* Free... */
00133 free(atm);
00134 free(met0);
00135 free(met1);
00136
00137 return EXIT_SUCCESS;
00138 }

```

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, o3, p0, p1, pref, ps, pt, pv, t, tt, u, v, w, z, zm, zref, zt;
00044
00045      int geopot, ip, it;
00046
00047      /* Check arguments... */
00048      if (argc < 4)
00049          ERRMSG("Give parameters: <ctl> <sample.tab> <metbase> <atm_in>");
00050
00051      /* Allocate... */
00052      ALLOC(atm, atm_t, 1);
00053      ALLOC(met0, met_t, 1);
00054      ALLOC(met1, met_t, 1);
00055
00056      /* Read control parameters... */
00057      read_ctl(argv[1], argc, argv, &ctl);
00058      geopot =
00059          (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GEOPOT", -1, "0", NULL);
00060
00061      /* Read atmospheric data... */
00062      if (!read_atm(argv[4], &ctl, atm))
00063          ERRMSG("Cannot open file!");
00064
00065      /* Create output file... */
00066      printf("Write meteorological data file: %s\n", argv[2]);
00067      if (!(out = fopen(argv[2], "w")))
00068          ERRMSG("Cannot create file!");
00069
00070      /* Write header... */
00071      fprintf(out,
00072          "# $1 = time [s]\n"
00073          "# $2 = altitude [km]\n"
00074          "# $3 = longitude [deg]\n"
00075          "# $4 = latitude [deg]\n"
00076          "# $5 = pressure [hPa]\n"
00077          "# $6 = temperature [K]\n"
00078          "# $7 = zonal wind [m/s]\n"
00079          "# $8 = meridional wind [m/s]\n"
00080          "# $9 = vertical wind [hPa/s]\n");
00081      fprintf(out,
00082          "# $10 = H2O volume mixing ratio [1]\n"
00083          "# $11 = O3 volume mixing ratio [1]\n"
00084          "# $12 = geopotential height [km]\n"
00085          "# $13 = potential vorticity [PVU]\n"
00086          "# $14 = surface pressure [hPa]\n"
00087          "# $15 = tropopause pressure [hPa]\n"
00088          "# $16 = tropopause geopotential height [km]\n"
00089          "# $17 = tropopause temperature [K]\n\n");
00090
00091      /* Loop over air parcels... */
00092      for (ip = 0; ip < atm->np; ip++) {
00093
00094          /* Get meteorological data... */
00095          get_met(&ctl, argv[3], atm->time[ip], &met0, &met1);
00096
00097          /* Set reference pressure for interpolation... */
00098          pref = atm->p[ip];
00099          if (geopot) {
00100              zref = Z(pref);
00101              p0 = met0->p[0];
00102              p1 = met0->p[met0->np - 1];
00103              for (it = 0; it < 24; it++) {

```

```

00104         pref = 0.5 * (p0 + p1);
00105         intpol_met_time(met0, met1, atm->time[ip], pref, atm->
lon[ip],
00106                         atm->lat[ip], NULL, NULL, &zm, NULL, NULL, NULL, NULL,
00107                         NULL, NULL, NULL);
00108         if (zref > zm || !gsl_finite(zm))
00109             p0 = pref;
00110         else
00111             p1 = pref;
00112     }
00113     pref = 0.5 * (p0 + p1);
00114 }
00115
00116 /* Interpolate meteorological data... */
00117 intpol_met_time(met0, met1, atm->time[ip], pref, atm->lon[ip],
00118                 atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &pv, &h2o,
00119                 &o3);
00120 intpol_met_time(met0, met1, atm->time[ip], pt, atm->lon[ip], atm->
lat[ip],
00121                 NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL, NULL);
00122
00123 /* Write data... */
00124 fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g\n",
00125         atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
00126         atm->p[ip], t, u, v, w, h2o, o3, z, pv, ps, pt, zt, tt);
00127 }
00128
00129 /* Close file... */
00130 fclose(out);
00131
00132 /* Free... */
00133 free(atm);
00134 free(met0);
00135 free(met1);
00136
00137 return EXIT_SUCCESS;
00138 }

```

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], ttm[NZ][NY],
00052         ztm[NZ][NY], tm[NZ][NY], um[NZ][NY], vm[NZ][NY], wm[NZ][NY], h2om[NZ][NY],
00053         pvm[NZ][NY], o3m[NZ][NY], zm[NZ][NY], z, z0, z1, dz, zt, tt, plev[NZ], ps,
00054         pt, t, u, v, w, pv, h2o, o3, lat, lat0, lat1, dlat, lats[NY];
00055
00056     static int i, ix, iy, iz, np[NZ][NY], npt[NZ][NY], ny, nz;
00057
00058     /* Allocate... */
00059     ALLOC(met, met_t, 1);
00060
00061     /* Check arguments... */
00062     if (argc < 4)
00063         ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00064
00065     /* Read control parameters... */
00066     read_ctl(argv[1], argc, argv, &ctl);
00067     z0 = scan_ctl(argv[1], argc, argv, "ZM_Z0", -1, "-999", NULL);
00068     z1 = scan_ctl(argv[1], argc, argv, "ZM_Z1", -1, "-999", NULL);
00069     dz = scan_ctl(argv[1], argc, argv, "ZM_DZ", -1, "-999", NULL);
00070     lat0 = scan_ctl(argv[1], argc, argv, "ZM_LAT0", -1, "-90", NULL);
00071     lat1 = scan_ctl(argv[1], argc, argv, "ZM_LAT1", -1, "90", NULL);
00072     dlat = scan_ctl(argv[1], argc, argv, "ZM_DLAT", -1, "-999", NULL);
00073
00074     /* Loop over files... */
00075     for (i = 3; i < argc; i++) {
00076
00077         /* Read meteorological data... */
00078         if (!read_met(&ctl, argv[i], met))
00079             continue;
00080
00081         /* Set vertical grid... */
00082         if (z0 < 0)
00083             z0 = Z(met->p[0]);
00084         if (z1 < 0)
00085             z1 = Z(met->p[met->np - 1]);
00086         nz = 0;
00087         if (dz < 0) {
00088             for (iz = 0; iz < met->np; iz++)
00089                 if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00090                     plev[nz] = met->p[iz];
00091                     if (++nz > NZ)
00092                         ERRMSG("Too many pressure levels!");
00093                 }
00094             } else
00095                 for (z = z0; z <= z1; z += dz) {
00096                     plev[nz] = P(z);
00097                     if (++nz > NZ)
00098                         ERRMSG("Too many pressure levels!");
00099                 }
00100
00101         /* Set horizontal grid... */
00102         if (dlat <= 0)
00103             dlat = fabs(met->lat[1] - met->lat[0]);
00104         ny = 0;
00105         if (lat0 < -90 && lat1 > 90) {
00106             lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00107             lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00108         }
00109         for (lat = lat0; lat <= lat1; lat += dlat) {
00110             lats[ny] = lat;
00111             if (++ny > NY)
00112                 ERRMSG("Too many latitudes!");
00113         }
00114
00115         /* Average... */
00116         for (ix = 0; ix < met->nx; ix++)
00117             for (iy = 0; iy < ny; iy++)
00118                 for (iz = 0; iz < nz; iz++) {
00119                     intpol_met_space(met, plev[iz], met->lon[ix], lats[iy], &ps,
00120                                     &pt, &z, &t, &u, &v, &w, &pv, &h2o, &o3);
00121                     intpol_met_space(met, pt, met->lon[ix], lats[iy], NULL, NULL,
00122                                     &zt, &tt, NULL, NULL, NULL, NULL, NULL, NULL);
00123                     timem[iz][iy] += met->time;
00124                     zm[iz][iy] += z;
00125                     tm[iz][iy] += t;
00126                     um[iz][iy] += u;
00127                     vm[iz][iy] += v;
00128                     wm[iz][iy] += w;
00129                     pvm[iz][iy] += pv;

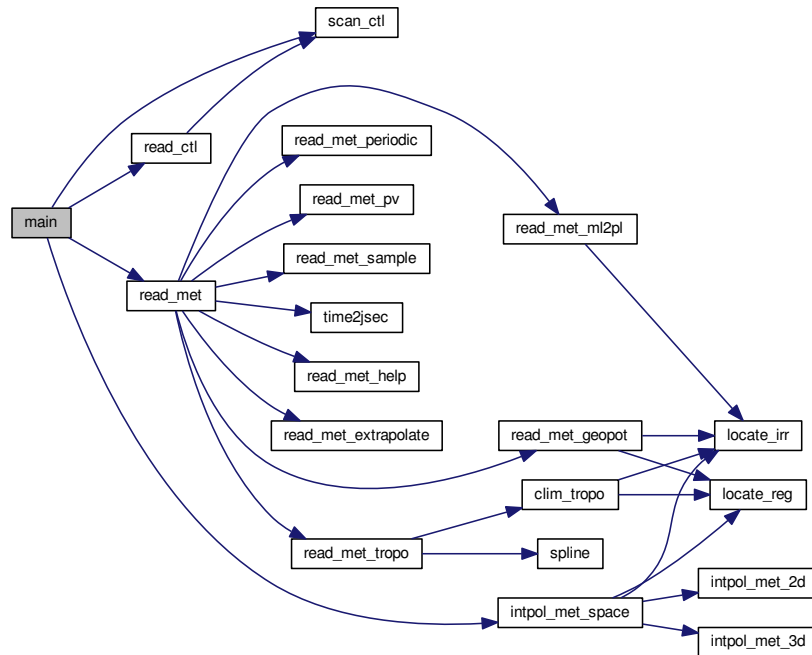
```

```

00130         h2om[iz][iy] += h2o;
00131         o3m[iz][iy] += o3;
00132         psm[iz][iy] += ps;
00133         if (gsl_finite(pt)) {
00134             ptm[iz][iy] += pt;
00135             ztm[iz][iy] += zt;
00136             ttm[iz][iy] += tt;
00137             npt[iz][iy]++;
00138         }
00139         np[iz][iy]++;
00140     }
00141 }
00142
00143 /* Create output file... */
00144 printf("Write meteorological data file: %s\n", argv[2]);
00145 if (! (out = fopen(argv[2], "w")))
00146     ERRMSG("Cannot create file!");
00147
00148 /* Write header... */
00149 fprintf(out,
00150         "# $1 = time [s]\n"
00151         "# $2 = altitude [km]\n"
00152         "# $3 = longitude [deg]\n"
00153         "# $4 = latitude [deg]\n"
00154         "# $5 = pressure [hPa]\n"
00155         "# $6 = temperature [K]\n"
00156         "# $7 = zonal wind [m/s]\n"
00157         "# $8 = meridional wind [m/s]\n" "# $9 = vertical wind [hPa/s]\n");
00158 fprintf(out,
00159         "# $10 = H2O volume mixing ratio [1]\n"
00160         "# $11 = O3 volume mixing ratio [1]\n"
00161         "# $12 = geopotential height [km]\n"
00162         "# $13 = potential vorticity [PVU]\n"
00163         "# $14 = surface pressure [hPa]\n"
00164         "# $15 = tropopause pressure [hPa]\n"
00165         "# $16 = tropopause geopotential height [km]\n"
00166         "# $17 = tropopause temperature [K]\n");
00167
00168 /* Write data... */
00169 for (iz = 0; iz < nz; iz++) {
00170     fprintf(out, "\n");
00171     for (iy = 0; iy < ny; iy++)
00172         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00173             timem[iz][iy] / np[iz][iy], Z(plev[iz]), 0.0, lats[iy],
00174             plev[iz], tm[iz][iy] / np[iz][iy], um[iz][iy] / np[iz][iy],
00175             vm[iz][iy] / np[iz][iy], wm[iz][iy] / np[iz][iy],
00176             h2om[iz][iy] / np[iz][iy], o3m[iz][iy] / np[iz][iy],
00177             zm[iz][iy] / np[iz][iy], pvm[iz][iy] / np[iz][iy],
00178             psm[iz][iy] / np[iz][iy], ptm[iz][iy] / npt[iz][iy],
00179             ztm[iz][iy] / npt[iz][iy], ttm[iz][iy] / npt[iz][iy]);
00180 }
00181
00182 /* Close file... */
00183 fclose(out);
00184
00185 /* Free... */
00186 free(met);
00187
00188 return EXIT_SUCCESS;
00189 }

```

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], ttm[NZ][NY],
00052  zm[NZ][NY], tm[NZ][NY], um[NZ][NY], vm[NZ][NY], wm[NZ][NY], h2om[NZ][NY],
00053  pvm[NZ][NY], o3m[NZ][NY], zm[NZ][NY], z, z0, z1, dz, zt, tt, plev[NZ], ps,
00054  pt, t, u, v, w, pv, h2o, o3, lat, lat0, lat1, dlat, lats[NY];
00055
00056  static int i, ix, iy, iz, np[NZ][NY], npt[NZ][NY], ny, nz;
00057
00058  /* Allocate... */
00059  ALLOC(met, met_t, 1);
00060
00061  /* Check arguments... */
00062  if (argc < 4)
00063      ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00064
00065  /* Read control parameters... */
00066  read_ctl(argv[1], argc, argv, &ctl);
00067  z0 = scan_ctl(argv[1], argc, argv, "ZM_Z0", -1, "-999", NULL);
00068  z1 = scan_ctl(argv[1], argc, argv, "ZM_Z1", -1, "-999", NULL);
00069  dz = scan_ctl(argv[1], argc, argv, "ZM_DZ", -1, "-999", NULL);
00070  lat0 = scan_ctl(argv[1], argc, argv, "ZM_LAT0", -1, "-90", NULL);
00071  lat1 = scan_ctl(argv[1], argc, argv, "ZM_LAT1", -1, "90", NULL);
00072  dlat = scan_ctl(argv[1], argc, argv, "ZM_DLAT", -1, "-999", NULL);
00073
00074  /* Loop over files... */
00075  for (i = 3; i < argc; i++) {
00076
00077      /* Read meteorological data... */
00078      if (!read_met(&ctl, argv[i], met))
00079          continue;
00080
00081      /* Set vertical grid... */
00082      if (z0 < 0)
00083          z0 = Z(met->p[0]);
00084      if (z1 < 0)
00085          z1 = Z(met->p[met->np - 1]);
00086      nz = 0;
00087      if (dz < 0) {
00088          for (iz = 0; iz < met->np; iz++)
00089              if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00090                  plev[nz] = met->p[iz];
00091                  if ((++nz) > NZ)
00092                      ERRMSG("Too many pressure levels!");
00093              }
00094      } else
00095          for (z = z0; z <= z1; z += dz) {
00096              plev[nz] = P(z);
00097              if ((++nz) > NZ)
00098                  ERRMSG("Too many pressure levels!");
00099          }
00100
00101      /* Set horizontal grid... */
00102      if (dlat <= 0)
00103          dlat = fabs(met->lat[1] - met->lat[0]);
00104      ny = 0;
00105      if (lat0 < -90 && lat1 > 90) {
00106          lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00107          lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00108      }
00109      for (lat = lat0; lat <= lat1; lat += dlat) {
00110          lats[ny] = lat;
00111          if ((++ny) > NY)
00112              ERRMSG("Too many latitudes!");
00113      }
00114
00115      /* Average... */
00116      for (ix = 0; ix < met->nx; ix++)
00117          for (iy = 0; iy < ny; iy++)
00118              for (iz = 0; iz < nz; iz++) {
00119                  intpol_met_space(met, plev[iz], met->lon[ix], lats[iy], &ps,
00120                  &pt, &z, &t, &u, &v, &w, &pv, &h2o, &o3);
00121                  intpol_met_space(met, pt, met->lon[ix], lats[iy], NULL, NULL,
00122                  &zt, &tt, NULL, NULL, NULL, NULL, NULL, NULL);
00123                  timem[iz][iy] += met->time;
00124                  zm[iz][iy] += z;
00125                  tm[iz][iy] += t;
00126                  um[iz][iy] += u;
00127                  vm[iz][iy] += v;
00128                  wm[iz][iy] += w;
00129                  pvm[iz][iy] += pv;
00130                  h2om[iz][iy] += h2o;
00131                  o3m[iz][iy] += o3;
00132                  psm[iz][iy] += ps;
00133                  if (gsl_finite(pt)) {

```

```

00134         ptm[iz][iy] += pt;
00135         ztm[iz][iy] += zt;
00136         ttm[iz][iy] += tt;
00137         npt[iz][iy]++;
00138     }
00139     np[iz][iy]++;
00140 }
00141 }
00142
00143 /* Create output file... */
00144 printf("Write meteorological data file: %s\n", argv[2]);
00145 if (!(out = fopen(argv[2], "w")))
00146     ERRMSG("Cannot create file!");
00147
00148 /* Write header... */
00149 fprintf(out,
00150         "# $1 = time [s]\n"
00151         "# $2 = altitude [km]\n"
00152         "# $3 = longitude [deg]\n"
00153         "# $4 = latitude [deg]\n"
00154         "# $5 = pressure [hPa]\n"
00155         "# $6 = temperature [K]\n"
00156         "# $7 = zonal wind [m/s]\n"
00157         "# $8 = meridional wind [m/s]\n" "# $9 = vertical wind [hPa/s]\n");
00158 fprintf(out,
00159         "# $10 = H2O volume mixing ratio [1]\n"
00160         "# $11 = O3 volume mixing ratio [1]\n"
00161         "# $12 = geopotential height [km]\n"
00162         "# $13 = potential vorticity [PVU]\n"
00163         "# $14 = surface pressure [hPa]\n"
00164         "# $15 = tropopause pressure [hPa]\n"
00165         "# $16 = tropopause geopotential height [km]\n"
00166         "# $17 = tropopause temperature [K]\n");
00167
00168 /* Write data... */
00169 for (iz = 0; iz < nz; iz++) {
00170     fprintf(out, "\n");
00171     for (iy = 0; iy < ny; iy++)
00172         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00173             timem[iz][iy] / np[iz][iy], Z(plev[iz]), 0.0, lats[iy],
00174             plev[iz], tm[iz][iy] / np[iz][iy], um[iz][iy] / np[iz][iy],
00175             vm[iz][iy] / np[iz][iy], wm[iz][iy] / np[iz][iy],
00176             h2om[iz][iy] / np[iz][iy], o3m[iz][iy] / np[iz][iy],
00177             zm[iz][iy] / np[iz][iy], pvm[iz][iy] / np[iz][iy],
00178             psm[iz][iy] / np[iz][iy], ptm[iz][iy] / npt[iz][iy],
00179             ztm[iz][iy] / npt[iz][iy], ttm[iz][iy] / npt[iz][iy]);
00180 }
00181
00182 /* Close file... */
00183 fclose(out);
00184
00185 /* Free... */
00186 free(met);
00187
00188 return EXIT_SUCCESS;
00189 }

```

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 [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

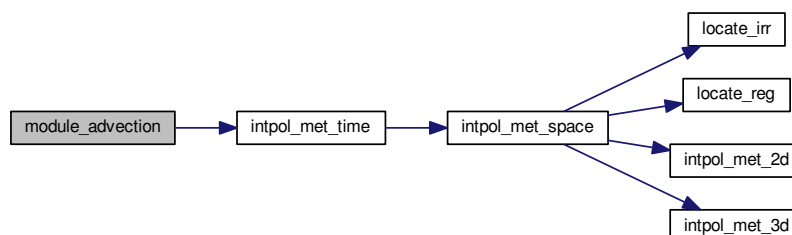
5.33.2.1 `void module_advection (met_t * met0, met_t * met1, atm_t * atm, double * dt)`

Calculate advection of air parcels.

Definition at line 419 of file [trac.c](#).

```
00423         {
00424
00425     #ifdef _OPENACC
00426     #pragma acc data present(met0,met1,atm,dt)
00427     #pragma acc parallel loop independent gang vector
00428     #else
00429     #pragma omp parallel for default(shared)
00430     #endif
00431     for (int ip = 0; ip < atm->np; ip++)
00432     if (dt[ip] != 0) {
00433
00434         double v[3], xm[3];
00435
00436         /* Interpolate meteorological data... */
00437         intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00438             atm->lon[ip], atm->lat[ip], NULL, NULL, NULL, NULL,
00439             &v[0], &v[1], &v[2], NULL, NULL, NULL);
00440
00441         /* Get position of the mid point... */
00442         xm[0] =
00443             atm->lon[ip] + DX2DEG(0.5 * dt[ip] * v[0] / 1000., atm->lat[ip]);
00444         xm[1] = atm->lat[ip] + DY2DEG(0.5 * dt[ip] * v[1] / 1000.);
00445         xm[2] = atm->p[ip] + 0.5 * dt[ip] * v[2];
00446
00447         /* Interpolate meteorological data for mid point... */
00448         intpol_met_time(met0, met1, atm->time[ip] + 0.5 * dt[ip],
00449             xm[2], xm[0], xm[1], NULL, NULL, NULL, NULL,
00450             &v[0], &v[1], &v[2], NULL, NULL, NULL);
00451
00452         /* Save new position... */
00453         atm->time[ip] += dt[ip];
00454         atm->lon[ip] += DX2DEG(dt[ip] * v[0] / 1000., xm[1]);
00455         atm->lat[ip] += DY2DEG(dt[ip] * v[1] / 1000.);
00456         atm->p[ip] += dt[ip] * v[2];
00457     }
00458 }
```

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.

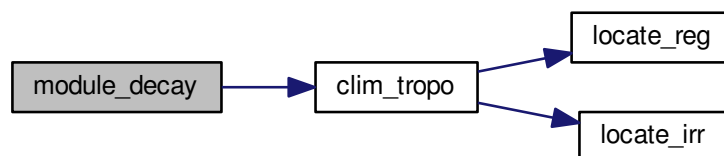
Definition at line 462 of file [trac.c](#).

```

00465         {
00466
00467     #ifdef _OPENACC
00468     #pragma acc data present(ctl,atm,dt)
00469     #pragma acc parallel loop independent gang vector
00470     #else
00471     #pragma omp parallel for default(shared)
00472     #endif
00473     for (int ip = 0; ip < atm->np; ip++)
00474     if (dt[ip] != 0) {
00475
00476         double p0, p1, pt, tdec, w;
00477
00478         /* Get tropopause pressure... */
00479         pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00480
00481         /* Get weighting factor... */
00482         p1 = pt * 0.866877899;
00483         p0 = pt / 0.866877899;
00484         if (atm->p[ip] > p0)
00485             w = 1;
00486         else if (atm->p[ip] < p1)
00487             w = 0;
00488         else
00489             w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00490
00491         /* Set lifetime... */
00492         tdec = w * ctl->tdec_trop + (1 - w) * ctl->tdec_strat;
00493
00494         /* Calculate exponential decay... */
00495         atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] / tdec);
00496     }
00497 }

```

Here is the call graph for this function:



5.33.2.3 void module_diffusion_init (void)

Initialize random number generator...

Definition at line 501 of file [trac.c](#).

```

00502     {
00503
00504     /* Initialize random number generator... */
00505     #ifdef _OPENACC
00506
00507     if (curandCreateGenerator(&rng, CURAND_RNG_PSEUDO_DEFAULT)

```

```

00508         != CURAND_STATUS_SUCCESS)
00509         ERRMSG("Cannot create random number generator!");
00510     if (curandSetStream(rng, (cudaStream_t) acc_get_cuda_stream(acc_async_sync))
00511         != CURAND_STATUS_SUCCESS)
00512         ERRMSG("Cannot set stream for random number generator!");
00513
00514 #else
00515
00516     gsl_rng_env_setup();
00517     if (omp_get_max_threads() > NTHREADS)
00518         ERRMSG("Too many threads!");
00519     for (int i = 0; i < NTHREADS; i++) {
00520         rng[i] = gsl_rng_alloc(gsl_rng_default);
00521         gsl_rng_set(rng[i], gsl_rng_default_seed + (long unsigned) i);
00522     }
00523
00524 #endif
00525 }

```

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 529 of file [trac.c](#).

```

00536     {
00537
00538 #ifdef _OPENACC
00539 #pragma acc data present(ctl,met0,met1,atm,cache,dt,rs)
00540 #pragma acc parallel loop independent gang vector
00541 #else
00542 #pragma omp parallel for default(shared)
00543 #endif
00544     for (int ip = 0; ip < atm->np; ip++)
00545         if (dt[ip] != 0) {
00546
00547             double u[16], v[16], w[16];
00548
00549             /* Get indices... */
00550             int ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
00551             int iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
00552             int iz = locate_irr(met0->p, met0->np, atm->p[ip]);
00553
00554             /* Caching of wind standard deviations... */
00555             if (cache->tsig[ix][iy][iz] != met0->time) {
00556
00557                 /* Collect local wind data... */
00558                 u[0] = met0->u[ix][iy][iz];
00559                 u[1] = met0->u[ix + 1][iy][iz];
00560                 u[2] = met0->u[ix][iy + 1][iz];
00561                 u[3] = met0->u[ix + 1][iy + 1][iz];
00562                 u[4] = met0->u[ix][iy][iz + 1];
00563                 u[5] = met0->u[ix + 1][iy][iz + 1];
00564                 u[6] = met0->u[ix][iy + 1][iz + 1];
00565                 u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00566
00567                 v[0] = met0->v[ix][iy][iz];
00568                 v[1] = met0->v[ix + 1][iy][iz];
00569                 v[2] = met0->v[ix][iy + 1][iz];
00570                 v[3] = met0->v[ix + 1][iy + 1][iz];
00571                 v[4] = met0->v[ix][iy][iz + 1];
00572                 v[5] = met0->v[ix + 1][iy][iz + 1];
00573                 v[6] = met0->v[ix][iy + 1][iz + 1];
00574                 v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00575
00576                 w[0] = met0->w[ix][iy][iz];
00577                 w[1] = met0->w[ix + 1][iy][iz];
00578                 w[2] = met0->w[ix][iy + 1][iz];
00579                 w[3] = met0->w[ix + 1][iy + 1][iz];
00580                 w[4] = met0->w[ix][iy][iz + 1];
00581                 w[5] = met0->w[ix + 1][iy][iz + 1];
00582                 w[6] = met0->w[ix][iy + 1][iz + 1];
00583                 w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00584
00585                 /* Collect local wind data... */
00586                 u[8] = met1->u[ix][iy][iz];
00587                 u[9] = met1->u[ix + 1][iy][iz];
00588                 u[10] = met1->u[ix][iy + 1][iz];
00589                 u[11] = met1->u[ix + 1][iy + 1][iz];

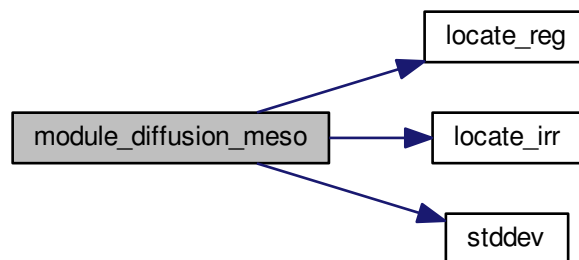
```

```

00590     u[12] = met1->u[ix][iy][iz + 1];
00591     u[13] = met1->u[ix + 1][iy][iz + 1];
00592     u[14] = met1->u[ix][iy + 1][iz + 1];
00593     u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00594
00595     v[8] = met1->v[ix][iy][iz];
00596     v[9] = met1->v[ix + 1][iy][iz];
00597     v[10] = met1->v[ix][iy + 1][iz];
00598     v[11] = met1->v[ix + 1][iy + 1][iz];
00599     v[12] = met1->v[ix][iy][iz + 1];
00600     v[13] = met1->v[ix + 1][iy][iz + 1];
00601     v[14] = met1->v[ix][iy + 1][iz + 1];
00602     v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00603
00604     w[8] = met1->w[ix][iy][iz];
00605     w[9] = met1->w[ix + 1][iy][iz];
00606     w[10] = met1->w[ix][iy + 1][iz];
00607     w[11] = met1->w[ix + 1][iy + 1][iz];
00608     w[12] = met1->w[ix][iy][iz + 1];
00609     w[13] = met1->w[ix + 1][iy][iz + 1];
00610     w[14] = met1->w[ix][iy + 1][iz + 1];
00611     w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00612
00613     /* Get standard deviations of local wind data... */
00614     cache->usig[ix][iy][iz] = (float) stddev(u, 16);
00615     cache->vsig[ix][iy][iz] = (float) stddev(v, 16);
00616     cache->wsig[ix][iy][iz] = (float) stddev(w, 16);
00617     cache->tsig[ix][iy][iz] = met0->time;
00618 }
00619
00620 /* Set temporal correlations for mesoscale fluctuations... */
00621 double r = 1 - 2 * fabs(dt[ip]) / ctl->dt_met;
00622 double r2 = sqrt(1 - r * r);
00623
00624 /* Calculate horizontal mesoscale wind fluctuations... */
00625 if (ctl->turb_mesox > 0) {
00626     cache->up[ip] = (float)
00627         (r * cache->up[ip]
00628          + r2 * rs[3 * ip] * ctl->turb_mesox * cache->usig[ix][iy][iz]);
00629     atm->lon[ip] += DX2DEG(cache->up[ip] * dt[ip] / 1000., atm->lat[ip]);
00630
00631     cache->vp[ip] = (float)
00632         (r * cache->vp[ip]
00633          + r2 * rs[3 * ip + 1] * ctl->turb_mesox * cache->vsig[ix][iy][iz]);
00634     atm->lat[ip] += DY2DEG(cache->vp[ip] * dt[ip] / 1000.);
00635 }
00636
00637 /* Calculate vertical mesoscale wind fluctuations... */
00638 if (ctl->turb_mesoz > 0) {
00639     cache->wp[ip] = (float)
00640         (r * cache->wp[ip]
00641          + r2 * rs[3 * ip + 2] * ctl->turb_mesoz * cache->wsig[ix][iy][iz]);
00642     atm->p[ip] += cache->wp[ip] * dt[ip];
00643 }
00644 }
00645 }

```

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 649 of file [trac.c](#).

```

00651         {
00652
00653 #ifdef _OPENACC
00654
00655 #pragma acc host_data use_device(rs)
00656     {
00657         if (curandGenerateNormalDouble(rng, rs, n, 0.0, 1.0)
00658             != CURAND_STATUS_SUCCESS)
00659             ERRMSG("Cannot create random numbers!");
00660     }
00661 #else
00662
00663 #pragma omp parallel for default(shared)
00664     for (size_t i = 0; i < n; ++i)
00665         rs[i] = gsl_ran_gaussian_ziggurat(rng[omp_get_thread_num()], 1.0);
00666 #endif
00667
00668 }
00669
00670 }
```

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

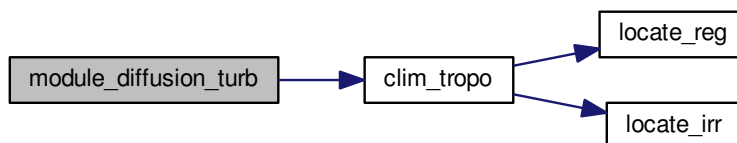
Calculate turbulent diffusion.

Definition at line 674 of file [trac.c](#).

```

00678         {
00679
00680 #ifdef _OPENACC
00681 #pragma acc data present(ctl,atm,dt,rs)
00682 #pragma acc parallel loop independent gang vector
00683 #else
00684 #pragma omp parallel for default(shared)
00685 #endif
00686     for (int ip = 0; ip < atm->np; ip++)
00687         if (dt[ip] != 0) {
00688
00689             double w;
00690
00691             /* Get tropopause pressure... */
00692             double pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00693
00694             /* Get weighting factor... */
00695             double p1 = pt * 0.866877899;
00696             double p0 = pt / 0.866877899;
00697             if (atm->p[ip] > p0)
00698                 w = 1;
00699             else if (atm->p[ip] < p1)
00700                 w = 0;
00701             else
00702                 w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00703
00704             /* Set diffusivity... */
00705             double dx = w * ctl->turb_dx_trop + (1 - w) * ctl->
turb_dx_strat;
00706             double dz = w * ctl->turb_dz_trop + (1 - w) * ctl->
turb_dz_strat;
00707
00708             /* Horizontal turbulent diffusion... */
00709             if (dx > 0) {
00710                 double sigma = sqrt(2.0 * dx * fabs(dt[ip]));
00711                 atm->lon[ip] += DX2DEG(rs[3 * ip] * sigma / 1000., atm->lat[ip]);
00712                 atm->lat[ip] += DY2DEG(rs[3 * ip + 1] * sigma / 1000.);
00713             }
00714
00715             /* Vertical turbulent diffusion... */
00716             if (dz > 0) {
00717                 double sigma = sqrt(2.0 * dz * fabs(dt[ip]));
00718                 atm->p[ip]
00719                     += DZ2DP(rs[3 * ip + 2] * sigma / 1000., atm->p[ip]);
00720             }
00721         }
00722 }
```

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.

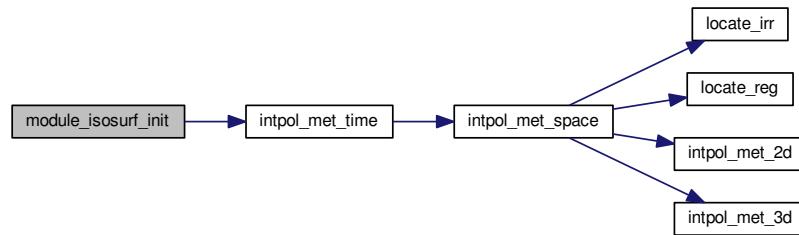
Definition at line 726 of file [trac.c](#).

```

00731         {
00732
00733     FILE *in;
00734
00735     char line[LEN];
00736
00737     double t;
00738
00739     /* Save pressure... */
00740     if (ctl->isosurf == 1)
00741         for (int ip = 0; ip < atm->np; ip++)
00742             cache->iso_var[ip] = atm->p[ip];
00743
00744     /* Save density... */
00745     else if (ctl->isosurf == 2)
00746         for (int ip = 0; ip < atm->np; ip++) {
00747             intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00748                             atm->lon[ip], atm->lat[ip], NULL, NULL, NULL,
00749                             &t, NULL, NULL, NULL, NULL, NULL, NULL);
00750             cache->iso_var[ip] = atm->p[ip] / t;
00751         }
00752
00753     /* Save potential temperature... */
00754     else if (ctl->isosurf == 3)
00755         for (int ip = 0; ip < atm->np; ip++) {
00756             intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00757                             atm->lon[ip], atm->lat[ip], NULL, NULL, NULL,
00758                             &t, NULL, NULL, NULL, NULL, NULL, NULL);
00759             cache->iso_var[ip] = THETA(atm->p[ip], t);
00760         }
00761
00762     /* Read balloon pressure data... */
00763     else if (ctl->isosurf == 4) {
00764
00765         /* Write info... */
00766         printf("Read balloon pressure data: %s\n", ctl->balloon);
00767
00768         /* Open file... */
00769         if (!(in = fopen(ctl->balloon, "r")))
00770             ERRMSG("Cannot open file!");
00771
00772         /* Read pressure time series... */
00773         while (fgets(line, LEN, in))
00774             if (sscanf(line, "%lg %lg", &(cache->iso_ts[cache->iso_n]),
00775                       &(cache->iso_ps[cache->iso_n])) == 2)
00776                 if (++cache->iso_n > NP)
00777                     ERRMSG("Too many data points!");
00778
00779         /* Check number of points... */
00780         if (cache->iso_n < 1)
00781             ERRMSG("Could not read any data!");
00782
00783         /* Close file... */
00784         fclose(in);
00785     }
00786 }

```

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.

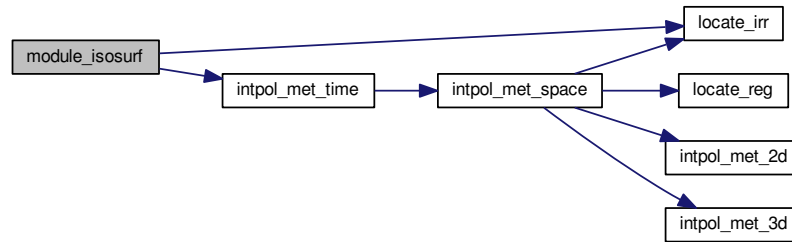
Definition at line 790 of file [trac.c](#).

```

00795         {
00796
00797     #ifdef _OPENACC
00798     #pragma acc data present(ctl,met0,met1,atm,cache)
00799     #pragma acc parallel loop independent gang vector
00800     #else
00801     #pragma omp parallel for default(shared)
00802     #endif
00803     for (int ip = 0; ip < atm->np; ip++) {
00804
00805         double t;
00806
00807         /* Restore pressure... */
00808         if (ctl->isosurf == 1)
00809             atm->p[ip] = cache->iso_var[ip];
00810
00811         /* Restore density... */
00812         else if (ctl->isosurf == 2) {
00813             intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
lon[ip],
00814                             atm->lat[ip], NULL, NULL, NULL, &t,
00815                             NULL, NULL, NULL, NULL, NULL, NULL);
00816             atm->p[ip] = cache->iso_var[ip] * t;
00817         }
00818
00819         /* Restore potential temperature... */
00820         else if (ctl->isosurf == 3) {
00821             intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
lon[ip],
00822                             atm->lat[ip], NULL, NULL, NULL, &t,
00823                             NULL, NULL, NULL, NULL, NULL, NULL);
00824             atm->p[ip] = 1000. * pow(cache->iso_var[ip] / t, -1. / 0.286);
00825         }
00826
00827         /* Interpolate pressure... */
00828         else if (ctl->isosurf == 4) {
00829             if (atm->time[ip] <= cache->iso_ts[0])
00830                 atm->p[ip] = cache->iso_ps[0];
00831             else if (atm->time[ip] >= cache->iso_ts[cache->iso_n - 1])
00832                 atm->p[ip] = cache->iso_ps[cache->iso_n - 1];
00833             else {
00834                 int idx = locate_irr(cache->iso_ts, cache->iso_n, atm->
time[ip]);
00835                 atm->p[ip] = LIN(cache->iso_ts[idx], cache->iso_ps[idx],
cache->iso_ts[idx + 1], cache->iso_ps[idx + 1],
00836                                atm->time[ip]);
00837             }
00838         }
00839     }
00840 }
00841 }

```

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 845 of file [trac.c](#).

```

00849         {
00850
00851     #ifdef _OPENACC
00852     #pragma acc data present(ctl,met0,met1,atm)
00853     #pragma acc parallel loop independent gang vector
00854     #else
00855     #pragma omp parallel for default(shared)
00856     #endif
00857     for (int ip = 0; ip < atm->np; ip++) {
00858
00859         double ps, pt, pv, t, u, v, w, h2o, o3, z;
00860
00861         /* Interpolate meteorological data... */
00862         intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
lon[ip],
00863                        atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &pv, &h2o,
00864                        &o3);
00865
00866         /* Set surface pressure... */
00867         if (ctl->qnt_ps >= 0)
00868             atm->q[ctl->qnt_ps][ip] = ps;
00869
00870         /* Set tropopause pressure... */
00871         if (ctl->qnt_pt >= 0)
00872             atm->q[ctl->qnt_pt][ip] = pt;
00873
00874         /* Set pressure... */
00875         if (ctl->qnt_p >= 0)
00876             atm->q[ctl->qnt_p][ip] = atm->p[ip];
00877
00878         /* Set geopotential height... */
00879         if (ctl->qnt_z >= 0)
00880             atm->q[ctl->qnt_z][ip] = z;
00881
00882         /* Set temperature... */
00883         if (ctl->qnt_t >= 0)
00884             atm->q[ctl->qnt_t][ip] = t;
00885
00886         /* Set zonal wind... */
00887         if (ctl->qnt_u >= 0)
00888             atm->q[ctl->qnt_u][ip] = u;
00889
00890         /* Set meridional wind... */
00891         if (ctl->qnt_v >= 0)
00892             atm->q[ctl->qnt_v][ip] = v;
00893
00894         /* Set vertical velocity... */
00895         if (ctl->qnt_w >= 0)
00896             atm->q[ctl->qnt_w][ip] = w;
00897
00898         /* Set water vapor vmr... */

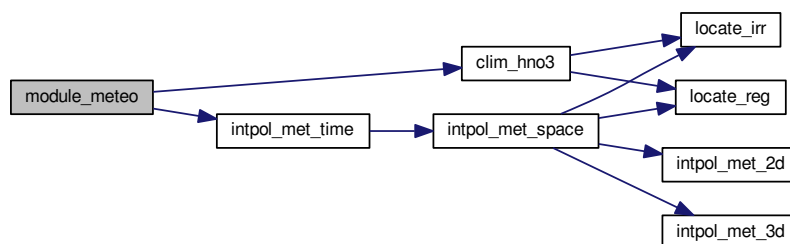
```

```

00899     if (ctl->qnt_h2o >= 0)
00900         atm->q[ctl->qnt_h2o][ip] = h2o;
00901
00902     /* Set ozone vmr... */
00903     if (ctl->qnt_o3 >= 0)
00904         atm->q[ctl->qnt_o3][ip] = o3;
00905
00906     /* Calculate horizontal wind... */
00907     if (ctl->qnt_vh >= 0)
00908         atm->q[ctl->qnt_vh][ip] = sqrt(u * u + v * v);
00909
00910     /* Calculate vertical velocity... */
00911     if (ctl->qnt_vz >= 0)
00912         atm->q[ctl->qnt_vz][ip] = -1e3 * H0 / atm->p[ip] * w;
00913
00914     /* Calculate potential temperature... */
00915     if (ctl->qnt_theta >= 0)
00916         atm->q[ctl->qnt_theta][ip] = THETA(atm->p[ip], t);
00917
00918     /* Set potential vorticity... */
00919     if (ctl->qnt_pv >= 0)
00920         atm->q[ctl->qnt_pv][ip] = pv;
00921
00922     /* Calculate T_ice (Marti and Mauersberger, 1993)... */
00923     if (ctl->qnt_tice >= 0)
00924         atm->q[ctl->qnt_tice][ip] =
00925             -2663.5 /
00926             (log10((ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] * 100.) -
00927              12.537);
00928
00929     /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
00930     if (ctl->qnt_tnat >= 0) {
00931         double p_hno3;
00932         if (ctl->psc_hno3 > 0)
00933             p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
00934         else
00935             p_hno3 = clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip])
00936                 * 1e-9 * atm->p[ip] / 1.333224;
00937         double p_h2o =
00938             (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
00939         double a = 0.009179 - 0.00088 * log10(p_h2o);
00940         double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
00941         double c = -11397.0 / a;
00942         double x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
00943         double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
00944         if (x1 > 0)
00945             atm->q[ctl->qnt_tnat][ip] = x1;
00946         if (x2 > 0)
00947             atm->q[ctl->qnt_tnat][ip] = x2;
00948     }
00949
00950     /* Calculate T_STS (mean of T_ice and T_NAT)... */
00951     if (ctl->qnt_tsts >= 0) {
00952         atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
00953                                         + atm->q[ctl->qnt_tnat][ip]);
00954     }
00955 }
00956 }

```

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.

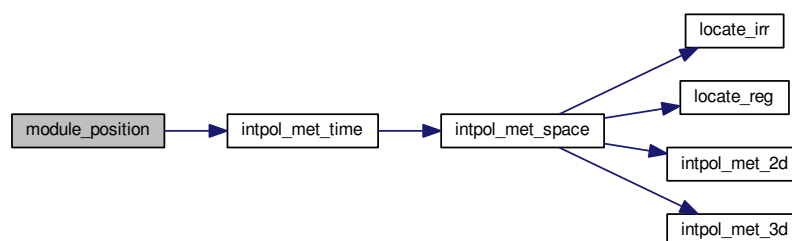
Definition at line 960 of file [trac.c](#).

```

00964         {
00965
00966 #ifdef _OPENACC
00967 #pragma acc data present (met0,met1,atm,dt)
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         if (dt[ip] != 0) {
00974
00975             double ps;
00976
00977             /* Calculate modulo... */
00978             atm->lon[ip] = FMOD(atm->lon[ip], 360.);
00979             atm->lat[ip] = FMOD(atm->lat[ip], 360.);
00980
00981             /* Check latitude... */
00982             while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
00983                 if (atm->lat[ip] > 90) {
00984                     atm->lat[ip] = 180 - atm->lat[ip];
00985                     atm->lon[ip] += 180;
00986                 }
00987                 if (atm->lat[ip] < -90) {
00988                     atm->lat[ip] = -180 - atm->lat[ip];
00989                     atm->lon[ip] += 180;
00990                 }
00991             }
00992
00993             /* Check longitude... */
00994             while (atm->lon[ip] < -180)
00995                 atm->lon[ip] += 360;
00996             while (atm->lon[ip] >= 180)
00997                 atm->lon[ip] -= 360;
00998
00999             /* Check pressure... */
01000             if (atm->p[ip] < met0->p[met0->np - 1])
01001                 atm->p[ip] = met0->p[met0->np - 1];
01002             else if (atm->p[ip] > 300.) {
01003                 intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
01004                               atm->lon[ip], atm->lat[ip], &ps, NULL, NULL, NULL,
01005                               NULL, NULL, NULL, NULL, NULL, NULL);
01006                 if (atm->p[ip] > ps)
01007                     atm->p[ip] = ps;
01008             }
01009         }
01010 }

```

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.

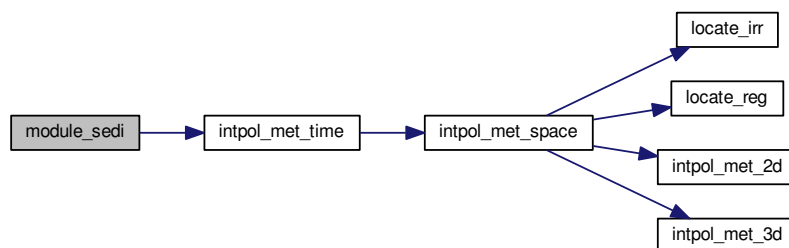
Definition at line 1014 of file [trac.c](#).

```

01019         {
01020
01021     #ifdef _OPENACC
01022     #pragma acc data present(ctl,met0,met1,atm,dt)
01023     #pragma acc parallel loop independent gang vector
01024     #else
01025     #pragma omp parallel for default(shared)
01026     #endif
01027     for (int ip = 0; ip < atm->np; ip++)
01028         if (dt[ip] != 0) {
01029
01030             /* Coefficients for Cunningham slip-flow correction (Kasten, 1968): */
01031             const double A = 1.249, B = 0.42, C = 0.87;
01032
01033             /* Average mass of an air molecule [kg/molec]: */
01034             const double m = 4.8096e-26;
01035
01036             double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
01037
01038             /* Convert units... */
01039             p = 100 * atm->p[ip];
01040             r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
01041             rho_p = atm->q[ctl->qnt_rho][ip];
01042
01043             /* Get temperature... */
01044             intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
lon[ip],
01045                             atm->lat[ip], NULL, NULL, NULL, &T,
01046                             NULL, NULL, NULL, NULL, NULL, NULL);
01047
01048             /* Density of dry air... */
01049             rho = p / (RA * T);
01050
01051             /* Dynamic viscosity of air... */
01052             eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
01053
01054             /* Thermal velocity of an air molecule... */
01055             v = sqrt(8 * KB * T / (M_PI * m));
01056
01057             /* Mean free path of an air molecule... */
01058             lambda = 2 * eta / (rho * v);
01059
01060             /* Knudsen number for air... */
01061             K = lambda / r_p;
01062
01063             /* Cunningham slip-flow correction... */
01064             G = 1 + K * (A + B * exp(-C / K));
01065
01066             /* Sedimentation (fall) velocity... */
01067             v_p = 2. * SQR(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;
01068
01069             /* Calculate pressure change... */
01070             atm->p[ip] += DZ2DP(v_p * dt[ip] / 1000., atm->p[ip]);
01071         }
01072     }

```

Here is the call graph for this function:



5.33.2.12 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 1076 of file [trac.c](#).

```

01082         {
01083
01084     char filename[2 * LEN];
01085
01086     double r;
01087
01088     int year, mon, day, hour, min, sec, updated = 0;
01089
01090     /* Get time... */
01091     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01092
01093     /* Write atmospheric data... */
01094     if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01095         if (!updated) {
01096 #ifdef _OPENACC
01097 #pragma acc update host(atm[:1])
01098 #endif
01099             updated = 1;
01100         }
01101         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.tab",
01102             dirname, ctl->atm_basename, year, mon, day, hour, min);
01103         write_atm(filename, ctl, atm, t);
01104     }
01105
01106     /* Write gridded data... */
01107     if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01108         if (!updated) {
01109 #ifdef _OPENACC
01110 #pragma acc update host(atm[:1])
01111 #endif
01112             updated = 1;
01113         }
01114         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01115             dirname, ctl->grid_basename, year, mon, day, hour, min);
01116         write_grid(filename, ctl, met0, met1, atm, t);
01117     }
01118
01119     /* Write CSI data... */
01120     if (ctl->csi_basename[0] != '-') {
01121         if (!updated) {
01122 #ifdef _OPENACC
01123 #pragma acc update host(atm[:1])
01124 #endif
01125             updated = 1;
01126         }
01127         sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01128         write_csi(filename, ctl, atm, t);
01129     }
01130
01131     /* Write ensemble data... */
01132     if (ctl->ens_basename[0] != '-') {
01133         if (!updated) {
01134 #ifdef _OPENACC
01135 #pragma acc update host(atm[:1])
01136 #endif
01137             updated = 1;
01138         }
01139         sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01140         write_ens(filename, ctl, atm, t);
01141     }
01142
01143     /* Write profile data... */
01144     if (ctl->prof_basename[0] != '-') {
01145         if (!updated) {
01146 #ifdef _OPENACC
01147 #pragma acc update host(atm[:1])
01148 #endif
01149             updated = 1;
01150         }
01151         sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01152         write_prof(filename, ctl, met0, met1, atm, t);
01153     }
01154
01155     /* Write station data... */
01156     if (ctl->stat_basename[0] != '-') {
01157         if (!updated) {
01158 #ifdef _OPENACC

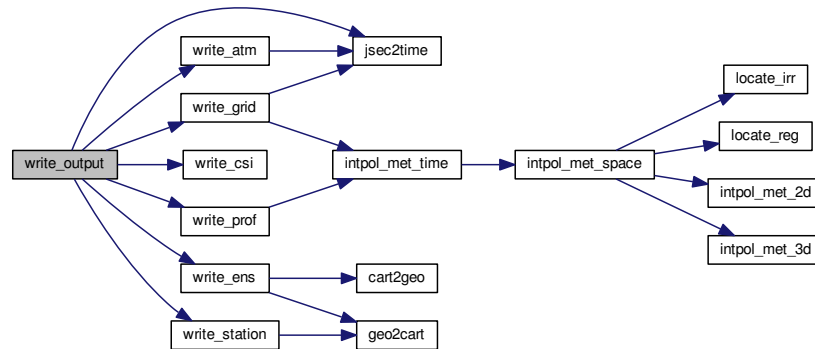
```

```

01159 #pragma acc update host (atm[:1])
01160 #endif
01161     updated = 1;
01162 }
01163     sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01164     write_station(filename, ctl, atm, t);
01165 }
01166 }

```

Here is the call graph for this function:



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

Definition at line 140 of file [trac.c](#).

```

00142     {
00143
00144     ctl_t ctl;
00145
00146     atm_t *atm;
00147
00148     cache_t *cache;
00149
00150     met_t *met0, *met1;
00151
00152     FILE *dirlist;
00153
00154     char dirname[LEN], filename[2 * LEN];
00155
00156     double *dt, *rs, t;
00157
00158     int ntask = -1, rank = 0, size = 1;
00159
00160     #ifdef MPI
00161     /* Initialize MPI... */
00162     MPI_Init(&argc, &argv);
00163     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00164     MPI_Comm_size(MPI_COMM_WORLD, &size);
00165     #endif
00166
00167     /* Check arguments... */
00168     if (argc < 5)
00169         ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00170
00171     /* Open directory list... */
00172     if (!(dirlist = fopen(argv[1], "r")))
00173         ERRMSG("Cannot open directory list!");
00174
00175     /* Loop over directories... */
00176     while (fscanf(dirlist, "%s", dirname) != EOF) {
00177
00178         /* MPI parallelization... */
00179         if ((++ntask) % size != rank)
00180             continue;

```

```

00181
00182 /* -----
00183     Initialize model run...
00184     ----- */
00185
00186 /* Set timers... */
00187 START_TIMER(TIMER_ZERO);
00188 START_TIMER(TIMER_TOTAL);
00189 START_TIMER(TIMER_INIT);
00190
00191 /* Allocate... */
00192 ALLOC(atm, atm_t, 1);
00193 ALLOC(cache, cache_t, 1);
00194 ALLOC(met0, met_t, 1);
00195 ALLOC(met1, met_t, 1);
00196 ALLOC(dt, double,
00197       NP);
00198 ALLOC(rs, double,
00199       3 * NP);
00200
00201 /* Read control parameters... */
00202 sprintf(filename, "%s/%s", dirname, argv[2]);
00203 read_ctl(filename, argc, argv, &ctl);
00204
00205 /* Read atmospheric data... */
00206 sprintf(filename, "%s/%s", dirname, argv[3]);
00207 if (!read_atm(filename, &ctl, atm))
00208     ERRMSG("Cannot open file!");
00209
00210 /* Copy to GPU... */
00211 #ifdef _OPENACC
00212 #pragma acc enter data copyin(ctl)
00213 #pragma acc enter data create(atm[:1], cache[:1], met0[:1], met1[:1], dt[:NP], rs[:3*NP])
00214 #pragma acc update device(atm[:1], cache[:1])
00215 #endif
00216
00217 /* Set start time... */
00218 if (ctl.direction == 1) {
00219     ctl.t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00220     if (ctl.t_stop > 1e99)
00221         ctl.t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00222 } else {
00223     ctl.t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00224     if (ctl.t_stop > 1e99)
00225         ctl.t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00226 }
00227
00228 /* Check time interval... */
00229 if (ctl.direction * (ctl.t_stop - ctl.t_start) <= 0)
00230     ERRMSG("Nothing to do!");
00231
00232 /* Round start time... */
00233 if (ctl.direction == 1)
00234     ctl.t_start = floor(ctl.t_start / ctl.dt_mod) * ctl.
dt_mod;
00235 else
00236     ctl.t_start = ceil(ctl.t_start / ctl.dt_mod) * ctl.
dt_mod;
00237
00238 /* Initialize random number generator... */
00239 module_diffusion_init();
00240
00241 /* Set timers... */
00242 STOP_TIMER(TIMER_INIT);
00243
00244 /* Initialize meteorological data... */
00245 START_TIMER(TIMER_INPUT);
00246 get_met(&ctl, argv[4], ctl.t_start, &met0, &met1);
00247 if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00248     WARN("Violation of CFL criterion! Check DT_MOD!");
00249 STOP_TIMER(TIMER_INPUT);
00250
00251 /* Initialize isosurface... */
00252 START_TIMER(TIMER_ISOSURF);
00253 if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00254     module_isosurf_init(&ctl, met0, met1, atm, cache);
00255 STOP_TIMER(TIMER_ISOSURF);
00256
00257 /* -----
00258     Loop over timesteps...
00259     ----- */
00260
00261 /* Loop over timesteps... */
00262 for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.
dt_mod;
00263      t += ctl.direction * ctl.dt_mod) {
00264

```

```

00265      /* Adjust length of final time step... */
00266      if (ctl.direction * (t - ctl.t_stop) > 0)
00267          t = ctl.t_stop;
00268
00269      /* Set time steps for air parcels... */
00270      #ifndef _OPENACC
00271      #pragma acc parallel loop independent gang vector present(ctl,atm,atm->time,dt)
00272      #endif
00273      for (int ip = 0; ip < atm->np; ip++) {
00274          double atmtime = atm->time[ip];
00275          double tstart = ctl.t_start;
00276          double tstop = ctl.t_stop;
00277          int dir = ctl.direction;
00278          if ((dir * (atmtime - tstart) >= 0 && dir * (atmtime - tstop) <= 0
00279              && dir * (atmtime - t) < 0))
00280              dt[ip] = t - atmtime;
00281          else
00282              dt[ip] = 0;
00283      }
00284
00285      /* Get meteorological data... */
00286      START_TIMER(TIMER_INPUT);
00287      if (t != ctl.t_start)
00288          get_met(&ctl, argv[4], t, &met0, &met1);
00289      STOP_TIMER(TIMER_INPUT);
00290
00291      /* Check initial position... */
00292      START_TIMER(TIMER_POSITION);
00293      module_position(met0, met1, atm, dt);
00294      STOP_TIMER(TIMER_POSITION);
00295
00296      /* Advection... */
00297      START_TIMER(TIMER_ADVECT);
00298      module_advection(met0, met1, atm, dt);
00299      STOP_TIMER(TIMER_ADVECT);
00300
00301      /* Turbulent diffusion... */
00302      START_TIMER(TIMER_DIFFTURB);
00303      if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
00304          || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0) {
00305          module_diffusion_rng(rs, 3 * (size_t) atm->np);
00306          module_diffusion_turb(&ctl, atm, dt, rs);
00307      }
00308      STOP_TIMER(TIMER_DIFFTURB);
00309
00310      /* Mesoscale diffusion... */
00311      START_TIMER(TIMER_DIFFMESO);
00312      if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0) {
00313          module_diffusion_rng(rs, 3 * (size_t) atm->np);
00314          module_diffusion_meso(&ctl, met0, met1, atm, cache, dt, rs);
00315      }
00316      STOP_TIMER(TIMER_DIFFMESO);
00317
00318      /* Sedimentation... */
00319      START_TIMER(TIMER_SEDI);
00320      if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0)
00321          module_sedi(&ctl, met0, met1, atm, dt);
00322      STOP_TIMER(TIMER_SEDI);
00323
00324      /* Isosurface... */
00325      START_TIMER(TIMER_ISOSURF);
00326      if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00327          module_isosurf(&ctl, met0, met1, atm, cache);
00328      STOP_TIMER(TIMER_ISOSURF);
00329
00330      /* Check final position... */
00331      START_TIMER(TIMER_POSITION);
00332      module_position(met0, met1, atm, dt);
00333      STOP_TIMER(TIMER_POSITION);
00334
00335      /* Interpolate meteorological data... */
00336      START_TIMER(TIMER_METEO);
00337      if (ctl.met_dt_out > 0
00338          && (ctl.met_dt_out < ctl.dt_mod || fmod(t, ctl.
00339 met_dt_out) == 0))
00339          module_meteo(&ctl, met0, met1, atm);
00340      STOP_TIMER(TIMER_METEO);
00341
00342      /* Decay of particle mass... */
00343      START_TIMER(TIMER_DECAY);
00344      if (ctl.tdec_trop > 0 && ctl.tdec_strat > 0 && ctl.
00345 qnt_m >= 0)
00345          module_decay(&ctl, atm, dt);
00346      STOP_TIMER(TIMER_DECAY);
00347
00348      /* Write output... */
00349      START_TIMER(TIMER_OUTPUT);

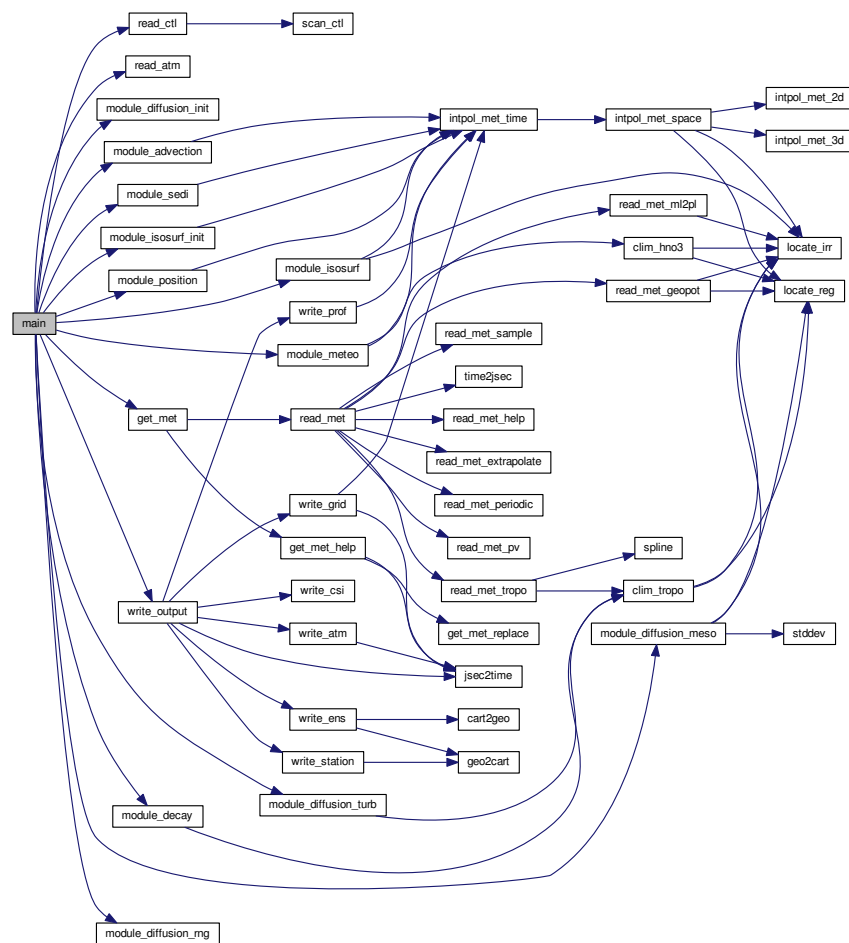
```

```

00350     write_output(dirname, &ctl, met0, met1, atm, t);
00351     STOP_TIMER(TIMER_OUTPUT);
00352 }
00353
00354 /* -----
00355     Finalize model run...
00356 ----- */
00357
00358 /* Report problem size... */
00359 printf("SIZE_NP = %d\n", atm->np);
00360 printf("SIZE_TASKS = %d\n", size);
00361 printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00362
00363 /* Report memory usage... */
00364 printf("MEMORY_ATM = %g MByte\n", sizeof(atm_t) / 1024. / 1024.);
00365 printf("MEMORY_CACHE = %g MByte\n", sizeof(cache_t) / 1024. / 1024.);
00366 printf("MEMORY_METEO = %g MByte\n", 2 * sizeof(met_t) / 1024. / 1024.);
00367 printf("MEMORY_DYNAMIC = %g MByte\n", (sizeof(met_t)
00368                                     + 4 * NP * sizeof(double)
00369                                     +
00370                                     EX * EY * EP * sizeof(float)) /
00371       1024. / 1024.);
00372 printf("MEMORY_STATIC = %g MByte\n", (EX * EY * sizeof(double)
00373                                     + EX * EY * EP * sizeof(float)
00374                                     + 4 * GX * GY * GZ * sizeof(double)
00375                                     + 2 * GX * GY * GZ * sizeof(int)
00376                                     + 2 * GX * GY * sizeof(double)
00377                                     +
00378                                     GX * GY * sizeof(int)) / 1024. /
00379       1024.);
00380
00381 /* Report timers... */
00382 STOP_TIMER(TIMER_ZERO);
00383 PRINT_TIMER(TIMER_INIT);
00384 PRINT_TIMER(TIMER_INPUT);
00385 PRINT_TIMER(TIMER_OUTPUT);
00386 PRINT_TIMER(TIMER_ADVECT);
00387 PRINT_TIMER(TIMER_DECAY);
00388 PRINT_TIMER(TIMER_DIFFMESO);
00389 PRINT_TIMER(TIMER_DIFFTURB);
00390 PRINT_TIMER(TIMER_ISOSURF);
00391 PRINT_TIMER(TIMER_METEO);
00392 PRINT_TIMER(TIMER_POSITION);
00393 PRINT_TIMER(TIMER_SEDI);
00394 STOP_TIMER(TIMER_TOTAL);
00395 PRINT_TIMER(TIMER_TOTAL);
00396
00397 /* Free... */
00398 free(atm);
00399 free(cache);
00400 free(met0);
00401 free(met1);
00402 free(dt);
00403 free(rs);
00404 #ifdef _OPENACC
00405 #pragma acc exit data delete(ctl, atm, cache, met0, met1, dt, rs)
00406 #endif
00407 }
00408
00409 #ifdef MPI
00410 /* Finalize MPI... */
00411 MPI_Finalize();
00412 #endif
00413
00414 return EXIT_SUCCESS;
00415 }

```

Here is the call graph for this function:



5.33.3 Variable Documentation

5.33.3.1 static gsl_rng * rng

Definition at line 41 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) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 #ifdef MPI
00028 #include "mpi.h"
00029 #endif
00030
00031 #ifdef _OPENACC
00032 #include "openacc.h"
00033 #include "curand.h"
00034 #endif
00035
00036 /* -----
00037 Global variables...
00038 ----- */
00039
00040 #ifdef _OPENACC
00041 curandGenerator_t rng;
00042 #else
00043 static gsl_rng *rng[NTHREADS];
00044 #endif
00045
00046 /* -----
00047 Functions...
00048 ----- */
00049
00051 void module_advection(
00052     met_t * met0,
00053     met_t * met1,
00054     atm_t * atm,
00055     double *dt);
00056
00058 void module_decay(
00059     ctl_t * ctl,
00060     atm_t * atm,
00061     double *dt);
00062
00064 void module_diffusion_init(
00065     void);
00066
00068 void module_diffusion_meso(
00069     ctl_t * ctl,
00070     met_t * met0,
00071     met_t * met1,
00072     atm_t * atm,
00073     cache_t * cache,
00074     double *dt,
00075     double *rs);
00076
00078 void module_diffusion_rng(
00079     double *rs,
00080     size_t n);
00081
00083 void module_diffusion_turb(
00084     ctl_t * ctl,
00085     atm_t * atm,
00086     double *dt,
00087     double *rs);
00088
00090 void module_isosurf_init(
00091     ctl_t * ctl,
00092     met_t * met0,
00093     met_t * met1,
00094     atm_t * atm,
00095     cache_t * cache);
00096
00098 void module_isosurf(
00099     ctl_t * ctl,
00100     met_t * met0,
00101     met_t * met1,
00102     atm_t * atm,
00103     cache_t * cache);
00104
00106 void module_meteo(
00107     ctl_t * ctl,
00108     met_t * met0,
00109     met_t * met1,
00110     atm_t * atm);
00111
00113 void module_position(
00114     met_t * met0,
00115     met_t * met1,
00116     atm_t * atm,
00117     double *dt);
```

```

00118
00120 void module_sedi(
00121     ctl_t * ctl,
00122     met_t * met0,
00123     met_t * met1,
00124     atm_t * atm,
00125     double *dt);
00126
00128 void write_output(
00129     const char *dirname,
00130     ctl_t * ctl,
00131     met_t * met0,
00132     met_t * met1,
00133     atm_t * atm,
00134     double t);
00135
00136 /* -----
00137     Main...
00138     ----- */
00139
00140 int main(
00141     int argc,
00142     char *argv[]) {
00143
00144     ctl_t ctl;
00145
00146     atm_t *atm;
00147
00148     cache_t *cache;
00149
00150     met_t *met0, *met1;
00151
00152     FILE *dirlist;
00153
00154     char dirname[LEN], filename[2 * LEN];
00155
00156     double *dt, *rs, t;
00157
00158     int ntask = -1, rank = 0, size = 1;
00159
00160 #ifdef MPI
00161     /* Initialize MPI... */
00162     MPI_Init(&argc, &argv);
00163     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00164     MPI_Comm_size(MPI_COMM_WORLD, &size);
00165 #endif
00166
00167     /* Check arguments... */
00168     if (argc < 5)
00169         ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00170
00171     /* Open directory list... */
00172     if (!(dirlist = fopen(argv[1], "r")))
00173         ERRMSG("Cannot open directory list!");
00174
00175     /* Loop over directories... */
00176     while (fscanf(dirlist, "%s", dirname) != EOF) {
00177
00178         /* MPI parallelization... */
00179         if ((++ntask) % size != rank)
00180             continue;
00181
00182         /* -----
00183             Initialize model run...
00184             ----- */
00185
00186         /* Set timers... */
00187         START_TIMER(TIMER_ZERO);
00188         START_TIMER(TIMER_TOTAL);
00189         START_TIMER(TIMER_INIT);
00190
00191         /* Allocate... */
00192         ALLOC(atm, atm_t, 1);
00193         ALLOC(cache, cache_t, 1);
00194         ALLOC(met0, met_t, 1);
00195         ALLOC(met1, met_t, 1);
00196         ALLOC(dt, double,
00197             NP);
00198         ALLOC(rs, double,
00199             3 * NP);
00200
00201         /* Read control parameters... */
00202         sprintf(filename, "%s/%s", dirname, argv[2]);
00203         read_ctl(filename, argc, argv, &ctl);
00204
00205         /* Read atmospheric data... */
00206         sprintf(filename, "%s/%s", dirname, argv[3]);

```

```

00207     if (!read_atm(filename, &ctl, atm))
00208         ERRMSG("Cannot open file!");
00209
00210     /* Copy to GPU... */
00211     #ifdef _OPENACC
00212     #pragma acc enter data copyin(ctl)
00213     #pragma acc enter data create(atm[:1],cache[:1],met0[:1],met1[:1],dt[:NP],rs[:3*NP])
00214     #pragma acc update device(atm[:1],cache[:1])
00215     #endif
00216
00217     /* Set start time... */
00218     if (ctl.direction == 1) {
00219         ctl.t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00220         if (ctl.t_stop > 1e99)
00221             ctl.t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00222     } else {
00223         ctl.t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00224         if (ctl.t_stop > 1e99)
00225             ctl.t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00226     }
00227
00228     /* Check time interval... */
00229     if (ctl.direction * (ctl.t_stop - ctl.t_start) <= 0)
00230         ERRMSG("Nothing to do!");
00231
00232     /* Round start time... */
00233     if (ctl.direction == 1)
00234         ctl.t_start = floor(ctl.t_start / ctl.dt_mod) * ctl.
dt_mod;
00235     else
00236         ctl.t_start = ceil(ctl.t_start / ctl.dt_mod) * ctl.
dt_mod;
00237
00238     /* Initialize random number generator... */
00239     module_diffusion_init();
00240
00241     /* Set timers... */
00242     STOP_TIMER(TIMER_INIT);
00243
00244     /* Initialize meteorological data... */
00245     START_TIMER(TIMER_INPUT);
00246     get_met(&ctl, argv[4], ctl.t_start, &met0, &met1);
00247     if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00248         WARN("Violation of CFL criterion! Check DT_MOD!");
00249     STOP_TIMER(TIMER_INPUT);
00250
00251     /* Initialize isosurface... */
00252     START_TIMER(TIMER_ISOSURF);
00253     if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00254         module_isosurf_init(&ctl, met0, met1, atm, cache);
00255     STOP_TIMER(TIMER_ISOSURF);
00256
00257     /* -----
00258     Loop over timesteps...
00259     ----- */
00260
00261     /* Loop over timesteps... */
00262     for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.
dt_mod;
00263         t += ctl.direction * ctl.dt_mod) {
00264
00265         /* Adjust length of final time step... */
00266         if (ctl.direction * (t - ctl.t_stop) > 0)
00267             t = ctl.t_stop;
00268
00269         /* Set time steps for air parcels... */
00270         #ifdef _OPENACC
00271         #pragma acc parallel loop independent gang vector present(ctl,atm,atm->time,dt)
00272         #endif
00273         for (int ip = 0; ip < atm->np; ip++) {
00274             double atmtime = atm->time[ip];
00275             double tstart = ctl.t_start;
00276             double tstop = ctl.t_stop;
00277             int dir = ctl.direction;
00278             if ((dir * (atmtime - tstart) >= 0 && dir * (atmtime - tstop) <= 0
&& dir * (atmtime - t) < 0))
00279                 dt[ip] = t - atmtime;
00280             else
00281                 dt[ip] = 0;
00282         }
00283
00284         /* Get meteorological data... */
00285         START_TIMER(TIMER_INPUT);
00286         if (t != ctl.t_start)
00287             get_met(&ctl, argv[4], t, &met0, &met1);
00288         STOP_TIMER(TIMER_INPUT);
00289
00290

```

```

00291      /* Check initial position... */
00292      START_TIMER(TIMER_POSITION);
00293      module_position(met0, met1, atm, dt);
00294      STOP_TIMER(TIMER_POSITION);
00295
00296      /* Advection... */
00297      START_TIMER(TIMER_ADVECT);
00298      module_advection(met0, met1, atm, dt);
00299      STOP_TIMER(TIMER_ADVECT);
00300
00301      /* Turbulent diffusion... */
00302      START_TIMER(TIMER_DIFFTURB);
00303      if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
00304          || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0) {
00305          module_diffusion_rng(rs, 3 * (size_t) atm->np);
00306          module_diffusion_turb(&ctl, atm, dt, rs);
00307      }
00308      STOP_TIMER(TIMER_DIFFTURB);
00309
00310      /* Mesoscale diffusion... */
00311      START_TIMER(TIMER_DIFFMESO);
00312      if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0) {
00313          module_diffusion_rng(rs, 3 * (size_t) atm->np);
00314          module_diffusion_meso(&ctl, met0, met1, atm, cache, dt, rs);
00315      }
00316      STOP_TIMER(TIMER_DIFFMESO);
00317
00318      /* Sedimentation... */
00319      START_TIMER(TIMER_SEDI);
00320      if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0)
00321          module_sedi(&ctl, met0, met1, atm, dt);
00322      STOP_TIMER(TIMER_SEDI);
00323
00324      /* Isosurface... */
00325      START_TIMER(TIMER_ISOSURF);
00326      if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00327          module_isosurf(&ctl, met0, met1, atm, cache);
00328      STOP_TIMER(TIMER_ISOSURF);
00329
00330      /* Check final position... */
00331      START_TIMER(TIMER_POSITION);
00332      module_position(met0, met1, atm, dt);
00333      STOP_TIMER(TIMER_POSITION);
00334
00335      /* Interpolate meteorological data... */
00336      START_TIMER(TIMER_METEO);
00337      if (ctl.met_dt_out > 0
00338          && (ctl.met_dt_out < ctl.dt_mod || fmod(t, ctl.
00339 met_dt_out) == 0))
00340          module_meteo(&ctl, met0, met1, atm);
00341      STOP_TIMER(TIMER_METEO);
00342
00343      /* Decay of particle mass... */
00344      START_TIMER(TIMER_DECAY);
00345      if (ctl.tdec_trop > 0 && ctl.tdec_strat > 0 && ctl.
00346 qnt_m >= 0)
00347          module_decay(&ctl, atm, dt);
00348      STOP_TIMER(TIMER_DECAY);
00349
00350      /* Write output... */
00351      START_TIMER(TIMER_OUTPUT);
00352      write_output(dirname, &ctl, met0, met1, atm, t);
00353      STOP_TIMER(TIMER_OUTPUT);
00354
00355      /* -----
00356      Finalize model run...
00357      ----- */
00358
00359      /* Report problem size... */
00360      printf("SIZE_NP = %d\n", atm->np);
00361      printf("SIZE_TASKS = %d\n", size);
00362      printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00363
00364      /* Report memory usage... */
00365      printf("MEMORY_ATM = %g MByte\n", sizeof(atm_t) / 1024. / 1024.);
00366      printf("MEMORY_CACHE = %g MByte\n", sizeof(cache_t) / 1024. / 1024.);
00367      printf("MEMORY_METEO = %g MByte\n", 2 * sizeof(met_t) / 1024. / 1024.);
00368      printf("MEMORY_DYNAMIC = %g MByte\n", (sizeof(met_t)
00369 + 4 * NP * sizeof(double)
00370 +
00371 EX * EY * EP * sizeof(float)) /
00372 1024. / 1024.);
00373      printf("MEMORY_STATIC = %g MByte\n", (EX * EY * sizeof(double)
00374 + EX * EY * EP * sizeof(float)
00375 + 4 * GX * GY * GZ * sizeof(double)
00376 + 2 * GX * GY * GZ * sizeof(int)

```

```

00376                                     + 2 * GX * GY * sizeof(double)
00377                                     +
00378                                     GX * GY * sizeof(int)) / 1024. /
00379     1024.);
00380
00381     /* Report timers... */
00382     STOP_TIMER(TIMER_ZERO);
00383     PRINT_TIMER(TIMER_INIT);
00384     PRINT_TIMER(TIMER_INPUT);
00385     PRINT_TIMER(TIMER_OUTPUT);
00386     PRINT_TIMER(TIMER_ADVECT);
00387     PRINT_TIMER(TIMER_DECAY);
00388     PRINT_TIMER(TIMER_DIFFMESO);
00389     PRINT_TIMER(TIMER_DIFFTURB);
00390     PRINT_TIMER(TIMER_ISOSURF);
00391     PRINT_TIMER(TIMER_METEO);
00392     PRINT_TIMER(TIMER_POSITION);
00393     PRINT_TIMER(TIMER_SEDI);
00394     STOP_TIMER(TIMER_TOTAL);
00395     PRINT_TIMER(TIMER_TOTAL);
00396
00397     /* Free... */
00398     free(atm);
00399     free(cache);
00400     free(met0);
00401     free(met1);
00402     free(dt);
00403     free(rs);
00404 #ifdef _OPENACC
00405 #pragma acc exit data delete(ctl, atm, cache, met0, met1, dt, rs)
00406 #endif
00407 }
00408
00409 #ifdef MPI
00410     /* Finalize MPI... */
00411     MPI_Finalize();
00412 #endif
00413
00414     return EXIT_SUCCESS;
00415 }
00416
00417 /*****
00418
00419 void module_advection(
00420     met_t * met0,
00421     met_t * met1,
00422     atm_t * atm,
00423     double *dt) {
00424
00425 #ifdef _OPENACC
00426 #pragma acc data present(met0, met1, atm, dt)
00427 #pragma acc parallel loop independent gang vector
00428 #else
00429 #pragma omp parallel for default(shared)
00430 #endif
00431     for (int ip = 0; ip < atm->np; ip++)
00432         if (dt[ip] != 0) {
00433
00434             double v[3], xm[3];
00435
00436             /* Interpolate meteorological data... */
00437             intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00438                             atm->lon[ip], atm->lat[ip], NULL, NULL, NULL,
00439                             &v[0], &v[1], &v[2], NULL, NULL, NULL);
00440
00441             /* Get position of the mid point... */
00442             xm[0] =
00443                 atm->lon[ip] + DX2DEG(0.5 * dt[ip] * v[0] / 1000., atm->lat[ip]);
00444             xm[1] = atm->lat[ip] + DY2DEG(0.5 * dt[ip] * v[1] / 1000.);
00445             xm[2] = atm->p[ip] + 0.5 * dt[ip] * v[2];
00446
00447             /* Interpolate meteorological data for mid point... */
00448             intpol_met_time(met0, met1, atm->time[ip] + 0.5 * dt[ip],
00449                             xm[2], xm[0], xm[1], NULL, NULL, NULL, NULL,
00450                             &v[0], &v[1], &v[2], NULL, NULL, NULL);
00451
00452             /* Save new position... */
00453             atm->time[ip] += dt[ip];
00454             atm->lon[ip] += DX2DEG(dt[ip] * v[0] / 1000., xm[1]);
00455             atm->lat[ip] += DY2DEG(dt[ip] * v[1] / 1000.);
00456             atm->p[ip] += dt[ip] * v[2];
00457         }
00458     }
00459
00460 /*****
00461
00462 void module_decay(

```

```

00463     ctl_t * ctl,
00464     atm_t * atm,
00465     double *dt) {
00466
00467     #ifdef _OPENACC
00468     #pragma acc data present(ctl,atm,dt)
00469     #pragma acc parallel loop independent gang vector
00470     #else
00471     #pragma omp parallel for default(shared)
00472     #endif
00473     for (int ip = 0; ip < atm->np; ip++)
00474         if (dt[ip] != 0) {
00475
00476             double p0, p1, pt, tdec, w;
00477
00478             /* Get tropopause pressure... */
00479             pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00480
00481             /* Get weighting factor... */
00482             p1 = pt * 0.866877899;
00483             p0 = pt / 0.866877899;
00484             if (atm->p[ip] > p0)
00485                 w = 1;
00486             else if (atm->p[ip] < p1)
00487                 w = 0;
00488             else
00489                 w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00490
00491             /* Set lifetime... */
00492             tdec = w * ctl->tdec_trop + (1 - w) * ctl->tdec_strat;
00493
00494             /* Calculate exponential decay... */
00495             atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] / tdec);
00496         }
00497 }
00498
00499 /*****
00500
00501 void module_diffusion_init(
00502     void) {
00503
00504     /* Initialize random number generator... */
00505     #ifdef _OPENACC
00506
00507         if (curandCreateGenerator(&rng, CURAND_RNG_PSEUDO_DEFAULT)
00508             != CURAND_STATUS_SUCCESS)
00509             ERRMSG("Cannot create random number generator!");
00510         if (curandSetStream(rng, (cudaStream_t) acc_get_cuda_stream(acc_async_sync))
00511             != CURAND_STATUS_SUCCESS)
00512             ERRMSG("Cannot set stream for random number generator!");
00513
00514     #else
00515
00516         gsl_rng_env_setup();
00517         if (omp_get_max_threads() > NTHREADS)
00518             ERRMSG("Too many threads!");
00519         for (int i = 0; i < NTHREADS; i++) {
00520             rng[i] = gsl_rng_alloc(gsl_rng_default);
00521             gsl_rng_set(rng[i], gsl_rng_default_seed + (long unsigned) i);
00522         }
00523
00524     #endif
00525 }
00526
00527 /*****
00528
00529 void module_diffusion_meso(
00530     ctl_t * ctl,
00531     met_t * met0,
00532     met_t * met1,
00533     atm_t * atm,
00534     cache_t * cache,
00535     double *dt,
00536     double *rs) {
00537
00538     #ifdef _OPENACC
00539     #pragma acc data present(ctl,met0,met1,atm,cache,dt,rs)
00540     #pragma acc parallel loop independent gang vector
00541     #else
00542     #pragma omp parallel for default(shared)
00543     #endif
00544     for (int ip = 0; ip < atm->np; ip++)
00545         if (dt[ip] != 0) {
00546
00547             double u[16], v[16], w[16];
00548
00549             /* Get indices... */

```

```

00550     int ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
00551     int iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
00552     int iz = locate_irr(met0->p, met0->np, atm->p[ip]);
00553
00554     /* Caching of wind standard deviations... */
00555     if (cache->tsig[ix][iy][iz] != met0->time) {
00556
00557         /* Collect local wind data... */
00558         u[0] = met0->u[ix][iy][iz];
00559         u[1] = met0->u[ix + 1][iy][iz];
00560         u[2] = met0->u[ix][iy + 1][iz];
00561         u[3] = met0->u[ix + 1][iy + 1][iz];
00562         u[4] = met0->u[ix][iy][iz + 1];
00563         u[5] = met0->u[ix + 1][iy][iz + 1];
00564         u[6] = met0->u[ix][iy + 1][iz + 1];
00565         u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00566
00567         v[0] = met0->v[ix][iy][iz];
00568         v[1] = met0->v[ix + 1][iy][iz];
00569         v[2] = met0->v[ix][iy + 1][iz];
00570         v[3] = met0->v[ix + 1][iy + 1][iz];
00571         v[4] = met0->v[ix][iy][iz + 1];
00572         v[5] = met0->v[ix + 1][iy][iz + 1];
00573         v[6] = met0->v[ix][iy + 1][iz + 1];
00574         v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00575
00576         w[0] = met0->w[ix][iy][iz];
00577         w[1] = met0->w[ix + 1][iy][iz];
00578         w[2] = met0->w[ix][iy + 1][iz];
00579         w[3] = met0->w[ix + 1][iy + 1][iz];
00580         w[4] = met0->w[ix][iy][iz + 1];
00581         w[5] = met0->w[ix + 1][iy][iz + 1];
00582         w[6] = met0->w[ix][iy + 1][iz + 1];
00583         w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00584
00585         /* Collect local wind data... */
00586         u[8] = met1->u[ix][iy][iz];
00587         u[9] = met1->u[ix + 1][iy][iz];
00588         u[10] = met1->u[ix][iy + 1][iz];
00589         u[11] = met1->u[ix + 1][iy + 1][iz];
00590         u[12] = met1->u[ix][iy][iz + 1];
00591         u[13] = met1->u[ix + 1][iy][iz + 1];
00592         u[14] = met1->u[ix][iy + 1][iz + 1];
00593         u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00594
00595         v[8] = met1->v[ix][iy][iz];
00596         v[9] = met1->v[ix + 1][iy][iz];
00597         v[10] = met1->v[ix][iy + 1][iz];
00598         v[11] = met1->v[ix + 1][iy + 1][iz];
00599         v[12] = met1->v[ix][iy][iz + 1];
00600         v[13] = met1->v[ix + 1][iy][iz + 1];
00601         v[14] = met1->v[ix][iy + 1][iz + 1];
00602         v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00603
00604         w[8] = met1->w[ix][iy][iz];
00605         w[9] = met1->w[ix + 1][iy][iz];
00606         w[10] = met1->w[ix][iy + 1][iz];
00607         w[11] = met1->w[ix + 1][iy + 1][iz];
00608         w[12] = met1->w[ix][iy][iz + 1];
00609         w[13] = met1->w[ix + 1][iy][iz + 1];
00610         w[14] = met1->w[ix][iy + 1][iz + 1];
00611         w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00612
00613         /* Get standard deviations of local wind data... */
00614         cache->usig[ix][iy][iz] = (float) stddev(u, 16);
00615         cache->vsig[ix][iy][iz] = (float) stddev(v, 16);
00616         cache->wsig[ix][iy][iz] = (float) stddev(w, 16);
00617         cache->tsig[ix][iy][iz] = met0->time;
00618     }
00619
00620     /* Set temporal correlations for mesoscale fluctuations... */
00621     double r = 1 - 2 * fabs(dt[ip]) / ctl->dt_met;
00622     double r2 = sqrt(1 - r * r);
00623
00624     /* Calculate horizontal mesoscale wind fluctuations... */
00625     if (ctl->turb_mesox > 0) {
00626         cache->up[ip] = (float)
00627             (r * cache->up[ip]
00628              + r2 * rs[3 * ip] * ctl->turb_mesox * cache->usig[ix][iy][iz]);
00629         atm->lon[ip] += DX2DEG(cache->up[ip] * dt[ip] / 1000., atm->lat[ip]);
00630
00631         cache->vp[ip] = (float)
00632             (r * cache->vp[ip]
00633              + r2 * rs[3 * ip + 1] * ctl->turb_mesox * cache->vsig[ix][iy][iz]);
00634         atm->lat[ip] += DY2DEG(cache->vp[ip] * dt[ip] / 1000.);
00635     }
00636

```

```

00637      /* Calculate vertical mesoscale wind fluctuations... */
00638      if (ctl->turb_mesoz > 0) {
00639          cache->wp[ip] = (float)
00640              (r * cache->wp[ip]
00641               + r2 * rs[3 * ip + 2] * ctl->turb_mesoz * cache->wsig[ix][iy][iz]);
00642          atm->p[ip] += cache->wp[ip] * dt[ip];
00643      }
00644  }
00645 }
00646
00647 /*****
00648
00649 void module_diffusion_rng(
00650     double *rs,
00651     size_t n) {
00652
00653 #ifdef _OPENACC
00654
00655 #pragma acc host_data use_device(rs)
00656 {
00657     if (curandGenerateNormalDouble(rng, rs, n, 0.0, 1.0)
00658         != CURAND_STATUS_SUCCESS)
00659         ERRMSG("Cannot create random numbers!");
00660 }
00661 #else
00662
00663 #pragma omp parallel for default(shared)
00664 for (size_t i = 0; i < n; ++i)
00665     rs[i] = gsl_ran_gaussian_ziggurat(rng[omp_get_thread_num()], 1.0);
00666 #endif
00667
00668 }
00669
00670
00671
00672 /*****
00673
00674 void module_diffusion_turb(
00675     ctl_t *ctl,
00676     atm_t *atm,
00677     double *dt,
00678     double *rs) {
00679
00680 #ifdef _OPENACC
00681 #pragma acc data present(ctl,atm,dt,rs)
00682 #pragma acc parallel loop independent gang vector
00683 #else
00684 #pragma omp parallel for default(shared)
00685 #endif
00686 for (int ip = 0; ip < atm->np; ip++)
00687     if (dt[ip] != 0) {
00688
00689         double w;
00690
00691         /* Get tropopause pressure... */
00692         double pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00693
00694         /* Get weighting factor... */
00695         double p1 = pt * 0.866877899;
00696         double p0 = pt / 0.866877899;
00697         if (atm->p[ip] > p0)
00698             w = 1;
00699         else if (atm->p[ip] < p1)
00700             w = 0;
00701         else
00702             w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00703
00704         /* Set diffusivity... */
00705         double dx = w * ctl->turb_dx_trop + (1 - w) * ctl->
00706 turb_dx_strat;
00707         double dz = w * ctl->turb_dz_trop + (1 - w) * ctl->
00708 turb_dz_strat;
00709
00710         /* Horizontal turbulent diffusion... */
00711         if (dx > 0) {
00712             double sigma = sqrt(2.0 * dx * fabs(dt[ip]));
00713             atm->lon[ip] += DX2DEG(rs[3 * ip] * sigma / 1000., atm->lat[ip]);
00714             atm->lat[ip] += DY2DEG(rs[3 * ip + 1] * sigma / 1000.);
00715         }
00716
00717         /* Vertical turbulent diffusion... */
00718         if (dz > 0) {
00719             double sigma = sqrt(2.0 * dz * fabs(dt[ip]));
00720             atm->p[ip]
00721                 += DZ2DP(rs[3 * ip + 2] * sigma / 1000., atm->p[ip]);
00722         }
00723     }
00724 }

```



```

00722 }
00723
00724 /*****
00725
00726 void module_isosurf_init(
00727     ctl_t * ctl,
00728     met_t * met0,
00729     met_t * met1,
00730     atm_t * atm,
00731     cache_t * cache) {
00732
00733     FILE *in;
00734
00735     char line[LEN];
00736
00737     double t;
00738
00739     /* Save pressure... */
00740     if (ctl->isosurf == 1)
00741         for (int ip = 0; ip < atm->np; ip++)
00742             cache->iso_var[ip] = atm->p[ip];
00743
00744     /* Save density... */
00745     else if (ctl->isosurf == 2)
00746         for (int ip = 0; ip < atm->np; ip++) {
00747             interpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00748                             atm->lon[ip], atm->lat[ip], NULL, NULL, NULL,
00749                             &t, NULL, NULL, NULL, NULL, NULL, NULL);
00750             cache->iso_var[ip] = atm->p[ip] / t;
00751         }
00752
00753     /* Save potential temperature... */
00754     else if (ctl->isosurf == 3)
00755         for (int ip = 0; ip < atm->np; ip++) {
00756             interpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00757                             atm->lon[ip], atm->lat[ip], NULL, NULL, NULL,
00758                             &t, NULL, NULL, NULL, NULL, NULL, NULL);
00759             cache->iso_var[ip] = THETA(atm->p[ip], t);
00760         }
00761
00762     /* Read balloon pressure data... */
00763     else if (ctl->isosurf == 4) {
00764
00765         /* Write info... */
00766         printf("Read balloon pressure data: %s\n", ctl->balloon);
00767
00768         /* Open file... */
00769         if (!(in = fopen(ctl->balloon, "r")))
00770             ERRMSG("Cannot open file!");
00771
00772         /* Read pressure time series... */
00773         while (fgets(line, LEN, in))
00774             if (sscanf(line, "%lg %lg", &(cache->iso_ts[cache->iso_n]),
00775                     &(cache->iso_ps[cache->iso_n])) == 2)
00776                 if ((++cache->iso_n) > NP)
00777                     ERRMSG("Too many data points!");
00778
00779         /* Check number of points... */
00780         if (cache->iso_n < 1)
00781             ERRMSG("Could not read any data!");
00782
00783         /* Close file... */
00784         fclose(in);
00785     }
00786 }
00787
00788 /*****
00789
00790 void module_isosurf(
00791     ctl_t * ctl,
00792     met_t * met0,
00793     met_t * met1,
00794     atm_t * atm,
00795     cache_t * cache) {
00796
00797     #ifdef _OPENACC
00798     #pragma acc data present(ctl,met0,met1,atm,cache)
00799     #pragma acc parallel loop independent gang vector
00800     #else
00801     #pragma omp parallel for default(shared)
00802     #endif
00803     for (int ip = 0; ip < atm->np; ip++) {
00804
00805         double t;
00806
00807         /* Restore pressure... */
00808         if (ctl->isosurf == 1)

```

```

00809     atm->p[ip] = cache->iso_var[ip];
00810
00811     /* Restore density... */
00812     else if (ctl->isosurf == 2) {
00813         intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
lon[ip],
00814                         atm->lat[ip], NULL, NULL, NULL, &t,
00815                         NULL, NULL, NULL, NULL, NULL, NULL);
00816         atm->p[ip] = cache->iso_var[ip] * t;
00817     }
00818
00819     /* Restore potential temperature... */
00820     else if (ctl->isosurf == 3) {
00821         intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
lon[ip],
00822                         atm->lat[ip], NULL, NULL, NULL, &t,
00823                         NULL, NULL, NULL, NULL, NULL, NULL);
00824         atm->p[ip] = 1000. * pow(cache->iso_var[ip] / t, -1. / 0.286);
00825     }
00826
00827     /* Interpolate pressure... */
00828     else if (ctl->isosurf == 4) {
00829         if (atm->time[ip] <= cache->iso_ts[0])
00830             atm->p[ip] = cache->iso_ps[0];
00831         else if (atm->time[ip] >= cache->iso_ts[cache->iso_n - 1])
00832             atm->p[ip] = cache->iso_ps[cache->iso_n - 1];
00833         else {
00834             int idx = locate_irr(cache->iso_ts, cache->iso_n, atm->
time[ip]);
00835             atm->p[ip] = LIN(cache->iso_ts[idx], cache->iso_ps[idx],
cache->iso_ts[idx + 1], cache->iso_ps[idx + 1],
00836                             atm->time[ip]);
00837         }
00838     }
00839 }
00840 }
00841 }
00842
00843 /*****
00844
00845 void module_meteo(
00846     ctl_t * ctl,
00847     met_t * met0,
00848     met_t * met1,
00849     atm_t * atm) {
00850
00851 #ifdef _OPENACC
00852 #pragma acc data present(ctl,met0,met1,atm)
00853 #pragma acc parallel loop independent gang vector
00854 #else
00855 #pragma omp parallel for default(shared)
00856 #endif
00857     for (int ip = 0; ip < atm->np; ip++) {
00858
00859         double ps, pt, pv, t, u, v, w, h2o, o3, z;
00860
00861         /* Interpolate meteorological data... */
00862         intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
lon[ip],
00863                         atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &pv, &h2o,
00864                         &o3);
00865
00866         /* Set surface pressure... */
00867         if (ctl->qnt_ps >= 0)
00868             atm->q[ctl->qnt_ps][ip] = ps;
00869
00870         /* Set tropopause pressure... */
00871         if (ctl->qnt_pt >= 0)
00872             atm->q[ctl->qnt_pt][ip] = pt;
00873
00874         /* Set pressure... */
00875         if (ctl->qnt_p >= 0)
00876             atm->q[ctl->qnt_p][ip] = atm->p[ip];
00877
00878         /* Set geopotential height... */
00879         if (ctl->qnt_z >= 0)
00880             atm->q[ctl->qnt_z][ip] = z;
00881
00882         /* Set temperature... */
00883         if (ctl->qnt_t >= 0)
00884             atm->q[ctl->qnt_t][ip] = t;
00885
00886         /* Set zonal wind... */
00887         if (ctl->qnt_u >= 0)
00888             atm->q[ctl->qnt_u][ip] = u;
00889
00890         /* Set meridional wind... */
00891         if (ctl->qnt_v >= 0)

```

```

00892     atm->q[ctl->qnt_v][ip] = v;
00893
00894     /* Set vertical velocity... */
00895     if (ctl->qnt_w >= 0)
00896         atm->q[ctl->qnt_w][ip] = w;
00897
00898     /* Set water vapor vmr... */
00899     if (ctl->qnt_h2o >= 0)
00900         atm->q[ctl->qnt_h2o][ip] = h2o;
00901
00902     /* Set ozone vmr... */
00903     if (ctl->qnt_o3 >= 0)
00904         atm->q[ctl->qnt_o3][ip] = o3;
00905
00906     /* Calculate horizontal wind... */
00907     if (ctl->qnt_vh >= 0)
00908         atm->q[ctl->qnt_vh][ip] = sqrt(u * u + v * v);
00909
00910     /* Calculate vertical velocity... */
00911     if (ctl->qnt_vz >= 0)
00912         atm->q[ctl->qnt_vz][ip] = -1e3 * H0 / atm->p[ip] * w;
00913
00914     /* Calculate potential temperature... */
00915     if (ctl->qnt_theta >= 0)
00916         atm->q[ctl->qnt_theta][ip] = THETA(atm->p[ip], t);
00917
00918     /* Set potential vorticity... */
00919     if (ctl->qnt_pv >= 0)
00920         atm->q[ctl->qnt_pv][ip] = pv;
00921
00922     /* Calculate T_ice (Marti and Mauersberger, 1993)... */
00923     if (ctl->qnt_tice >= 0)
00924         atm->q[ctl->qnt_tice][ip] =
00925             -2663.5 /
00926             (log10((ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] * 100.) -
00927              12.537);
00928
00929     /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
00930     if (ctl->qnt_tnat >= 0) {
00931         double p_hno3;
00932         if (ctl->psc_hno3 > 0)
00933             p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
00934         else
00935             p_hno3 = clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip])
00936                 * 1e-9 * atm->p[ip] / 1.333224;
00937         double p_h2o =
00938             (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
00939         double a = 0.009179 - 0.00088 * log10(p_h2o);
00940         double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
00941         double c = -11397.0 / a;
00942         double x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
00943         double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
00944         if (x1 > 0)
00945             atm->q[ctl->qnt_tnat][ip] = x1;
00946         if (x2 > 0)
00947             atm->q[ctl->qnt_tnat][ip] = x2;
00948     }
00949
00950     /* Calculate T_STS (mean of T_ice and T_NAT)... */
00951     if (ctl->qnt_tsts >= 0) {
00952         atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
00953                                         + atm->q[ctl->qnt_tnat][ip]);
00954     }
00955 }
00956 }
00957
00958 /*****
00959
00960 void module_position(
00961     met_t * met0,
00962     met_t * met1,
00963     atm_t * atm,
00964     double *dt) {
00965
00966 #ifdef _OPENACC
00967 #pragma acc data present(met0,met1,atm,dt)
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         if (dt[ip] != 0) {
00974
00975             double ps;
00976
00977             /* Calculate modulo... */
00978             atm->lon[ip] = FMOD(atm->lon[ip], 360.);

```

```

00979     atm->lat[ip] = FMOD(atm->lat[ip], 360.);
00980
00981     /* Check latitude... */
00982     while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
00983         if (atm->lat[ip] > 90) {
00984             atm->lat[ip] = 180 - atm->lat[ip];
00985             atm->lon[ip] += 180;
00986         }
00987         if (atm->lat[ip] < -90) {
00988             atm->lat[ip] = -180 - atm->lat[ip];
00989             atm->lon[ip] += 180;
00990         }
00991     }
00992
00993     /* Check longitude... */
00994     while (atm->lon[ip] < -180)
00995         atm->lon[ip] += 360;
00996     while (atm->lon[ip] >= 180)
00997         atm->lon[ip] -= 360;
00998
00999     /* Check pressure... */
01000     if (atm->p[ip] < met0->p[met0->np - 1])
01001         atm->p[ip] = met0->p[met0->np - 1];
01002     else if (atm->p[ip] > 300.) {
01003         intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
01004             atm->lon[ip], atm->lat[ip], &ps, NULL, NULL, NULL,
01005             NULL, NULL, NULL, NULL, NULL, NULL);
01006         if (atm->p[ip] > ps)
01007             atm->p[ip] = ps;
01008     }
01009 }
01010 }
01011
01012 /*****
01013
01014 void module_sedi(
01015     ctl_t * ctl,
01016     met_t * met0,
01017     met_t * met1,
01018     atm_t * atm,
01019     double *dt) {
01020
01021 #ifdef _OPENACC
01022 #pragma acc data present(ctl,met0,met1,atm,dt)
01023 #pragma acc parallel loop independent gang vector
01024 #else
01025 #pragma omp parallel for default(shared)
01026 #endif
01027     for (int ip = 0; ip < atm->np; ip++)
01028         if (dt[ip] != 0) {
01029
01030             /* Coefficients for Cunningham slip-flow correction (Kasten, 1968): */
01031             const double A = 1.249, B = 0.42, C = 0.87;
01032
01033             /* Average mass of an air molecule [kg/molec]: */
01034             const double m = 4.8096e-26;
01035
01036             double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
01037
01038             /* Convert units... */
01039             p = 100 * atm->p[ip];
01040             r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
01041             rho_p = atm->q[ctl->qnt_rho][ip];
01042
01043             /* Get temperature... */
01044             intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
lon[ip],
01045                 atm->lat[ip], NULL, NULL, NULL, &T,
01046                 NULL, NULL, NULL, NULL, NULL, NULL);
01047
01048             /* Density of dry air... */
01049             rho = p / (RA * T);
01050
01051             /* Dynamic viscosity of air... */
01052             eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
01053
01054             /* Thermal velocity of an air molecule... */
01055             v = sqrt(8 * KB * T / (M_PI * m));
01056
01057             /* Mean free path of an air molecule... */
01058             lambda = 2 * eta / (rho * v);
01059
01060             /* Knudsen number for air... */
01061             K = lambda / r_p;
01062
01063             /* Cunningham slip-flow correction... */
01064             G = 1 + K * (A + B * exp(-C / K));

```

```

01065
01066     /* Sedimentation (fall) velocity... */
01067     v_p = 2. * SQR(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;
01068
01069     /* Calculate pressure change... */
01070     atm->p[ip] += DZ2DP(v_p * dt[ip] / 1000., atm->p[ip]);
01071 }
01072 }
01073
01074 /*****
01075
01076 void write_output(
01077     const char *dirname,
01078     ctl_t * ctl,
01079     met_t * met0,
01080     met_t * met1,
01081     atm_t * atm,
01082     double t) {
01083
01084     char filename[2 * LEN];
01085
01086     double r;
01087
01088     int year, mon, day, hour, min, sec, updated = 0;
01089
01090     /* Get time... */
01091     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01092
01093     /* Write atmospheric data... */
01094     if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01095         if (!updated) {
01096             #ifdef _OPENACC
01097             #pragma acc update host (atm[:1])
01098             #endif
01099             updated = 1;
01100         }
01101         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.tab",
01102             dirname, ctl->atm_basename, year, mon, day, hour, min);
01103         write_atm(filename, ctl, atm, t);
01104     }
01105
01106     /* Write gridded data... */
01107     if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01108         if (!updated) {
01109             #ifdef _OPENACC
01110             #pragma acc update host (atm[:1])
01111             #endif
01112             updated = 1;
01113         }
01114         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01115             dirname, ctl->grid_basename, year, mon, day, hour, min);
01116         write_grid(filename, ctl, met0, met1, atm, t);
01117     }
01118
01119     /* Write CSI data... */
01120     if (ctl->csi_basename[0] != '-') {
01121         if (!updated) {
01122             #ifdef _OPENACC
01123             #pragma acc update host (atm[:1])
01124             #endif
01125             updated = 1;
01126         }
01127         sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01128         write_csi(filename, ctl, atm, t);
01129     }
01130
01131     /* Write ensemble data... */
01132     if (ctl->ens_basename[0] != '-') {
01133         if (!updated) {
01134             #ifdef _OPENACC
01135             #pragma acc update host (atm[:1])
01136             #endif
01137             updated = 1;
01138         }
01139         sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01140         write_ens(filename, ctl, atm, t);
01141     }
01142
01143     /* Write profile data... */
01144     if (ctl->prof_basename[0] != '-') {
01145         if (!updated) {
01146             #ifdef _OPENACC
01147             #pragma acc update host (atm[:1])
01148             #endif
01149             updated = 1;
01150         }
01151         sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);

```

```

01152     write_prof(filename, ctl, met0, met1, atm, t);
01153 }
01154
01155 /* Write station data... */
01156 if (ctl->stat_basename[0] != '-') {
01157     if (!updated) {
01158 #ifdef _OPENACC
01159 #pragma acc update host(atm[:1])
01160 #endif
01161         updated = 1;
01162     }
01163     sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01164     write_station(filename, ctl, atm, t);
01165 }
01166 }

```

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 [322](#) of file [tropo.c](#).

```

00326     {
00327
00328     int varid;
00329
00330     NC(nc_inq_varid(ncid, varname, &varid));
00331     NC(nc_put_att_text(ncid, varid, attrname, strlen(text), text));
00332 }

```

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

Definition at line 51 of file [tropo.c](#).

```

00053         {
00054
00055     ctl_t ctl;
00056
00057     met_t *met;
00058
00059     static double pt[NX * NY], qt[NX * NY], zt[NX * NY], tt[NX * NY], lon, lon0,
00060         lon1, lons[NX], dlon, lat, lat0, lat1, lats[NY], dlat;
00061
00062     static int init, i, ix, iy, nx, ny, nt, ncid, dims[3], timid, lonid, latid,
00063         clppid, clpqid, clptid, clpzid, dynpid, dynqid, dyntid, dynzid, wmolpid,
00064         wmolqid, wmoltid, wmolzid, wmo2pid, wmo2qid, wmo2tid, wmo2zid;
00065
00066     static size_t count[10], start[10];
00067
00068     /* Allocate... */
00069     ALLOC(met, met_t, 1);
00070
00071     /* Check arguments... */
00072     if (argc < 4)
00073         ERRMSG("Give parameters: <ctl> <tropo.nc> <met0> [ <met1> ... ]");
00074
00075     /* Read control parameters... */
00076     read_ctl(argv[1], argc, argv, &ctl);
00077     lon0 = scan_ctl(argv[1], argc, argv, "MAP_LON0", -1, "-180", NULL);
00078     lon1 = scan_ctl(argv[1], argc, argv, "MAP_LON1", -1, "180", NULL);
00079     dlon = scan_ctl(argv[1], argc, argv, "MAP_DLON", -1, "-999", NULL);
00080     lat0 = scan_ctl(argv[1], argc, argv, "MAP_LAT0", -1, "-90", NULL);
00081     lat1 = scan_ctl(argv[1], argc, argv, "MAP_LAT1", -1, "90", NULL);
00082     dlat = scan_ctl(argv[1], argc, argv, "MAP_DLAT", -1, "-999", NULL);
00083
00084     /* Loop over files... */
00085     for (i = 3; i < argc; i++) {
00086
00087         /* Read meteorological data... */
00088         ctl.met_tropo = 0;
00089         if (!read_met(&ctl, argv[i], met))
00090             continue;
00091
00092         /* Set horizontal grid... */
00093         if (!init) {
00094             init = 1;
00095
00096             /* get grid... */
00097             if (dlon <= 0)
00098                 dlon = fabs(met->lon[1] - met->lon[0]);
00099             if (dlat <= 0)
00100                 dlat = fabs(met->lat[1] - met->lat[0]);
00101             if (lon0 < -360 && lon1 > 360) {
00102                 lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00103                 lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00104             }
00105             nx = ny = 0;
00106             for (lon = lon0; lon <= lon1; lon += dlon) {
00107                 lons[nx] = lon;
00108                 if ((++nx) > NX)
00109                     ERRMSG("Too many longitudes!");
00110             }
00111             if (lat0 < -90 && lat1 > 90) {
00112                 lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00113                 lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00114             }
00115             for (lat = lat0; lat <= lat1; lat += dlat) {
00116                 lats[ny] = lat;
00117                 if ((++ny) > NY)
00118                     ERRMSG("Too many latitudes!");
00119             }
00120
00121             /* Create netCDF file... */
00122             printf("Write tropopause data file: %s\n", argv[2]);
00123             NC(nc_create(argv[2], NC_CLOBBER, &ncid));
00124
00125             /* Create dimensions... */
00126             NC(nc_def_dim(ncid, "time", (size_t) NC_UNLIMITED, &dims[0]));
00127             NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[1]));
00128             NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[2]));
00129
00130             /* Create variables... */
00131             NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));
00132             NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[1], &latid));

```

```

00133     NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[2], &lonid));
00134     NC(nc_def_var(ncid, "clp_z", NC_FLOAT, 3, &dims[0], &clpzid));
00135     NC(nc_def_var(ncid, "clp_p", NC_FLOAT, 3, &dims[0], &clppid));
00136     NC(nc_def_var(ncid, "clp_t", NC_FLOAT, 3, &dims[0], &clptid));
00137     NC(nc_def_var(ncid, "clp_q", NC_FLOAT, 3, &dims[0], &clpqid));
00138     NC(nc_def_var(ncid, "dyn_z", NC_FLOAT, 3, &dims[0], &dynzid));
00139     NC(nc_def_var(ncid, "dyn_p", NC_FLOAT, 3, &dims[0], &dynpid));
00140     NC(nc_def_var(ncid, "dyn_t", NC_FLOAT, 3, &dims[0], &dyntid));
00141     NC(nc_def_var(ncid, "dyn_q", NC_FLOAT, 3, &dims[0], &dynqid));
00142     NC(nc_def_var(ncid, "wmo_1st_z", NC_FLOAT, 3, &dims[0], &wmo1zid));
00143     NC(nc_def_var(ncid, "wmo_1st_p", NC_FLOAT, 3, &dims[0], &wmo1pid));
00144     NC(nc_def_var(ncid, "wmo_1st_t", NC_FLOAT, 3, &dims[0], &wmo1tid));
00145     NC(nc_def_var(ncid, "wmo_1st_q", NC_FLOAT, 3, &dims[0], &wmo1qid));
00146     NC(nc_def_var(ncid, "wmo_2nd_z", NC_FLOAT, 3, &dims[0], &wmo2zid));
00147     NC(nc_def_var(ncid, "wmo_2nd_p", NC_FLOAT, 3, &dims[0], &wmo2pid));
00148     NC(nc_def_var(ncid, "wmo_2nd_t", NC_FLOAT, 3, &dims[0], &wmo2tid));
00149     NC(nc_def_var(ncid, "wmo_2nd_q", NC_FLOAT, 3, &dims[0], &wmo2qid));
00150
00151     /* Set attributes... */
00152     add_text_attribute(ncid, "time", "units", "s");
00153     add_text_attribute(ncid, "time", "long_name",
00154         "seconds since 2000-01-01, 00:00 UTC");
00155     add_text_attribute(ncid, "lon", "units", "degrees_east");
00156     add_text_attribute(ncid, "lon", "long_name", "longitude");
00157     add_text_attribute(ncid, "lat", "units", "degrees_north");
00158     add_text_attribute(ncid, "lat", "long_name", "latitude");
00159
00160     add_text_attribute(ncid, "clp_z", "units", "km");
00161     add_text_attribute(ncid, "clp_z", "long_name", "cold point height");
00162     add_text_attribute(ncid, "clp_p", "units", "hPa");
00163     add_text_attribute(ncid, "clp_p", "long_name", "cold point pressure");
00164     add_text_attribute(ncid, "clp_t", "units", "K");
00165     add_text_attribute(ncid, "clp_t", "long_name",
00166         "cold point temperature");
00167     add_text_attribute(ncid, "clp_q", "units", "ppv");
00168     add_text_attribute(ncid, "clp_q", "long_name",
00169         "cold point water vapor");
00170
00171     add_text_attribute(ncid, "dyn_z", "units", "km");
00172     add_text_attribute(ncid, "dyn_z", "long_name",
00173         "dynamical tropopause height");
00174     add_text_attribute(ncid, "dyn_p", "units", "hPa");
00175     add_text_attribute(ncid, "dyn_p", "long_name",
00176         "dynamical tropopause pressure");
00177     add_text_attribute(ncid, "dyn_t", "units", "K");
00178     add_text_attribute(ncid, "dyn_t", "long_name",
00179         "dynamical tropopause temperature");
00180     add_text_attribute(ncid, "dyn_q", "units", "ppv");
00181     add_text_attribute(ncid, "dyn_q", "long_name",
00182         "dynamical tropopause water vapor");
00183
00184     add_text_attribute(ncid, "wmo_1st_z", "units", "km");
00185     add_text_attribute(ncid, "wmo_1st_z", "long_name",
00186         "WMO 1st tropopause height");
00187     add_text_attribute(ncid, "wmo_1st_p", "units", "hPa");
00188     add_text_attribute(ncid, "wmo_1st_p", "long_name",
00189         "WMO 1st tropopause pressure");
00190     add_text_attribute(ncid, "wmo_1st_t", "units", "K");
00191     add_text_attribute(ncid, "wmo_1st_t", "long_name",
00192         "WMO 1st tropopause temperature");
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     add_text_attribute(ncid, "wmo_2nd_z", "units", "km");
00198     add_text_attribute(ncid, "wmo_2nd_z", "long_name",
00199         "WMO 2nd tropopause height");
00200     add_text_attribute(ncid, "wmo_2nd_p", "units", "hPa");
00201     add_text_attribute(ncid, "wmo_2nd_p", "long_name",
00202         "WMO 2nd tropopause pressure");
00203     add_text_attribute(ncid, "wmo_2nd_t", "units", "K");
00204     add_text_attribute(ncid, "wmo_2nd_t", "long_name",
00205         "WMO 2nd tropopause temperature");
00206     add_text_attribute(ncid, "wmo_2nd_q", "units", "ppv");
00207     add_text_attribute(ncid, "wmo_2nd_q", "long_name",
00208         "WMO 2nd tropopause water vapor");
00209
00210     /* End definition... */
00211     NC(nc_enddef(ncid));
00212
00213     /* Write longitude and latitude... */
00214     NC(nc_put_var_double(ncid, latid, lats));
00215     NC(nc_put_var_double(ncid, lonid, lons));
00216 }
00217
00218 /* Write time... */
00219 start[0] = (size_t) nt;

```



```

00220     count[0] = 1;
00221     start[1] = 0;
00222     count[1] = (size_t) ny;
00223     start[2] = 0;
00224     count[2] = (size_t) nx;
00225     NC(nc_put_vara_double(ncid, timid, start, count, &met->time));
00226
00227     /* Get cold point... */
00228     ctl.met_tropo = 2;
00229     read_met_tropo(&ctl, met);
00230 #pragma omp parallel for default(shared) private(ix,iy)
00231     for (ix = 0; ix < nx; ix++)
00232         for (iy = 0; iy < ny; iy++) {
00233             intpol_met_space(met, 100, lons[ix], lats[iy], NULL,
00234                             &pt[iy * nx + ix], NULL, NULL, NULL,
00235                             NULL, NULL, NULL, NULL, NULL);
00236             intpol_met_space(met, pt[iy * nx + ix], lons[ix], lats[iy],
00237                             NULL, NULL, &zt[iy * nx + ix], &tt[iy * nx + ix],
00238                             NULL, NULL, NULL, NULL, &qt[iy * nx + ix], NULL);
00239         }
00240
00241     /* Write data... */
00242     NC(nc_put_vara_double(ncid, clpzid, start, count, zt));
00243     NC(nc_put_vara_double(ncid, clppid, start, count, pt));
00244     NC(nc_put_vara_double(ncid, clptid, start, count, tt));
00245     NC(nc_put_vara_double(ncid, clpqid, start, count, qt));
00246
00247     /* Get dynamical tropopause... */
00248     ctl.met_tropo = 5;
00249     read_met_tropo(&ctl, met);
00250 #pragma omp parallel for default(shared) private(ix,iy)
00251     for (ix = 0; ix < nx; ix++)
00252         for (iy = 0; iy < ny; iy++) {
00253             intpol_met_space(met, 100, lons[ix], lats[iy], NULL,
00254                             &pt[iy * nx + ix], NULL, NULL, NULL,
00255                             NULL, NULL, NULL, NULL, NULL);
00256             intpol_met_space(met, pt[iy * nx + ix], lons[ix], lats[iy],
00257                             NULL, NULL, &zt[iy * nx + ix], &tt[iy * nx + ix],
00258                             NULL, NULL, NULL, NULL, &qt[iy * nx + ix], NULL);
00259         }
00260
00261     /* Write data... */
00262     NC(nc_put_vara_double(ncid, dynzid, start, count, zt));
00263     NC(nc_put_vara_double(ncid, dynpid, start, count, pt));
00264     NC(nc_put_vara_double(ncid, dyntid, start, count, tt));
00265     NC(nc_put_vara_double(ncid, dynqid, start, count, qt));
00266
00267     /* Get WMO 1st tropopause... */
00268     ctl.met_tropo = 3;
00269     read_met_tropo(&ctl, met);
00270 #pragma omp parallel for default(shared) private(ix,iy)
00271     for (ix = 0; ix < nx; ix++)
00272         for (iy = 0; iy < ny; iy++) {
00273             intpol_met_space(met, 100, lons[ix], lats[iy], NULL,
00274                             &pt[iy * nx + ix], NULL, NULL, NULL,
00275                             NULL, NULL, NULL, NULL, NULL);
00276             intpol_met_space(met, pt[iy * nx + ix], lons[ix], lats[iy],
00277                             NULL, NULL, &zt[iy * nx + ix], &tt[iy * nx + ix],
00278                             NULL, NULL, NULL, NULL, &qt[iy * nx + ix], NULL);
00279         }
00280
00281     /* Write data... */
00282     NC(nc_put_vara_double(ncid, wmolzid, start, count, zt));
00283     NC(nc_put_vara_double(ncid, wmolpid, start, count, pt));
00284     NC(nc_put_vara_double(ncid, wmoltid, start, count, tt));
00285     NC(nc_put_vara_double(ncid, wmolqid, start, count, qt));
00286
00287     /* Get WMO 2nd tropopause... */
00288     ctl.met_tropo = 4;
00289     read_met_tropo(&ctl, met);
00290 #pragma omp parallel for default(shared) private(ix,iy)
00291     for (ix = 0; ix < nx; ix++)
00292         for (iy = 0; iy < ny; iy++) {
00293             intpol_met_space(met, 100, lons[ix], lats[iy], NULL,
00294                             &pt[iy * nx + ix], NULL, NULL, NULL,
00295                             NULL, NULL, NULL, NULL, NULL);
00296             intpol_met_space(met, pt[iy * nx + ix], lons[ix], lats[iy],
00297                             NULL, NULL, &zt[iy * nx + ix], &tt[iy * nx + ix],
00298                             NULL, NULL, NULL, NULL, &qt[iy * nx + ix], NULL);
00299         }
00300
00301     /* Write data... */
00302     NC(nc_put_vara_double(ncid, wmo2zid, start, count, zt));
00303     NC(nc_put_vara_double(ncid, wmo2pid, start, count, pt));
00304     NC(nc_put_vara_double(ncid, wmo2tid, start, count, tt));
00305     NC(nc_put_vara_double(ncid, wmo2qid, start, count, qt));
00306

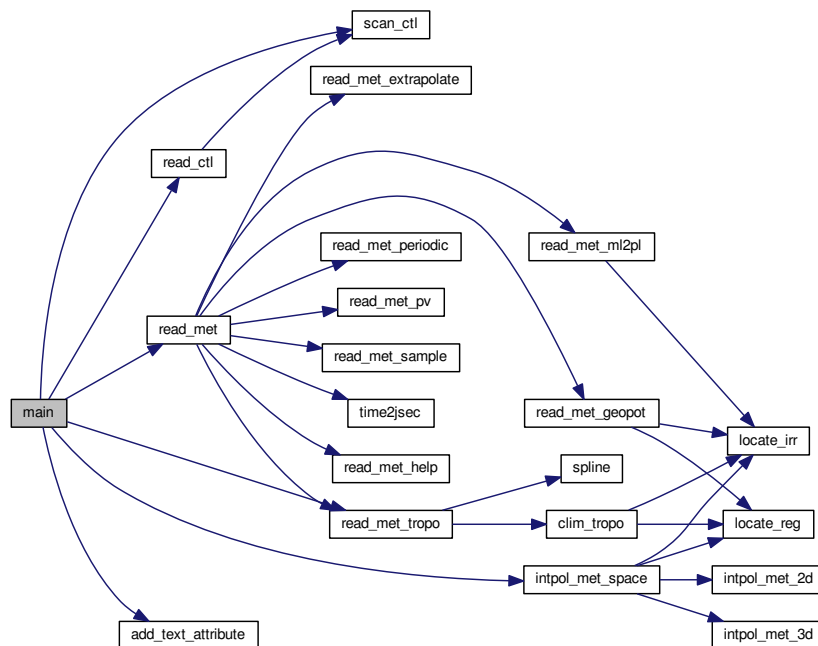
```

```

00307     /* Increment time step counter... */
00308     nt++;
00309 }
00310
00311 /* Close file... */
00312 NC(nc_close(ncid));
00313
00314 /* Free... */
00315 free(met);
00316
00317 return EXIT_SUCCESS;
00318 }

```

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  Dimensions...
00029  ----- */
00030

```

```

00032 #define NX 1441
00033
00035 #define NY 721
00036
00037 /* -----
00038     Functions...
00039     ----- */
00040
00041 void add_text_attribute(
00042     int ncid,
00043     char *varname,
00044     char *attrname,
00045     char *text);
00046
00047 /* -----
00048     Main...
00049     ----- */
00050
00051 int main(
00052     int argc,
00053     char *argv[]) {
00054
00055     ctl_t ctl;
00056
00057     met_t *met;
00058
00059     static double pt[NX * NY], qt[NX * NY], zt[NX * NY], tt[NX * NY], lon, lon0,
00060         lon1, lons[NX], dlon, lat, lat0, lat1, lats[NY], dlat;
00061
00062     static int init, i, ix, iy, nx, ny, nt, ncid, dims[3], timid, lonid, latid,
00063         clppid, clpqid, clptid, clpzid, dynpid, dynqid, dyntid, dynzid, wmolpid,
00064         wmolqid, wmoltid, wmolzid, wmo2pid, wmo2qid, wmo2tid, wmo2zid;
00065
00066     static size_t count[10], start[10];
00067
00068     /* Allocate... */
00069     ALLOC(met, met_t, 1);
00070
00071     /* Check arguments... */
00072     if (argc < 4)
00073         ERRMSG("Give parameters: <ctl> <tropo.nc> <met0> [ <met1> ... ]");
00074
00075     /* Read control parameters... */
00076     read_ctl(argv[1], argc, argv, &ctl);
00077     lon0 = scan_ctl(argv[1], argc, argv, "MAP_LON0", -1, "-180", NULL);
00078     lon1 = scan_ctl(argv[1], argc, argv, "MAP_LON1", -1, "180", NULL);
00079     dlon = scan_ctl(argv[1], argc, argv, "MAP_DLON", -1, "-999", NULL);
00080     lat0 = scan_ctl(argv[1], argc, argv, "MAP_LAT0", -1, "-90", NULL);
00081     lat1 = scan_ctl(argv[1], argc, argv, "MAP_LAT1", -1, "90", NULL);
00082     dlat = scan_ctl(argv[1], argc, argv, "MAP_DLAT", -1, "-999", NULL);
00083
00084     /* Loop over files... */
00085     for (i = 3; i < argc; i++) {
00086
00087         /* Read meteorological data... */
00088         ctl.met_tropo = 0;
00089         if (!read_met(&ctl, argv[i], met))
00090             continue;
00091
00092         /* Set horizontal grid... */
00093         if (!init) {
00094             init = 1;
00095
00096             /* get grid... */
00097             if (dlon <= 0)
00098                 dlon = fabs(met->lon[1] - met->lon[0]);
00099             if (dlat <= 0)
00100                 dlat = fabs(met->lat[1] - met->lat[0]);
00101             if (lon0 < -360 && lon1 > 360) {
00102                 lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00103                 lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00104             }
00105             nx = ny = 0;
00106             for (lon = lon0; lon <= lon1; lon += dlon) {
00107                 lons[nx] = lon;
00108                 if ((++nx) > NX)
00109                     ERRMSG("Too many longitudes!");
00110             }
00111             if (lat0 < -90 && lat1 > 90) {
00112                 lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00113                 lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00114             }
00115             for (lat = lat0; lat <= lat1; lat += dlat) {
00116                 lats[ny] = lat;
00117                 if ((++ny) > NY)
00118                     ERRMSG("Too many latitudes!");
00119             }

```

```

00120
00121      /* Create netCDF file... */
00122      printf("Write tropopause data file: %s\n", argv[2]);
00123      NC(nc_create(argv[2], NC_CLOBBER, &ncid));
00124
00125      /* Create dimensions... */
00126      NC(nc_def_dim(ncid, "time", (size_t) NC_UNLIMITED, &dims[0]));
00127      NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[1]));
00128      NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[2]));
00129
00130      /* Create variables... */
00131      NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));
00132      NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[1], &latid));
00133      NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[2], &lonid));
00134      NC(nc_def_var(ncid, "clp_z", NC_FLOAT, 3, &dims[0], &clpzid));
00135      NC(nc_def_var(ncid, "clp_p", NC_FLOAT, 3, &dims[0], &clppid));
00136      NC(nc_def_var(ncid, "clp_t", NC_FLOAT, 3, &dims[0], &clptid));
00137      NC(nc_def_var(ncid, "clp_q", NC_FLOAT, 3, &dims[0], &clpqid));
00138      NC(nc_def_var(ncid, "dyn_z", NC_FLOAT, 3, &dims[0], &dynzid));
00139      NC(nc_def_var(ncid, "dyn_p", NC_FLOAT, 3, &dims[0], &dynpid));
00140      NC(nc_def_var(ncid, "dyn_t", NC_FLOAT, 3, &dims[0], &dyntid));
00141      NC(nc_def_var(ncid, "dyn_q", NC_FLOAT, 3, &dims[0], &dynqid));
00142      NC(nc_def_var(ncid, "wmo_1st_z", NC_FLOAT, 3, &dims[0], &wmo1zid));
00143      NC(nc_def_var(ncid, "wmo_1st_p", NC_FLOAT, 3, &dims[0], &wmo1pid));
00144      NC(nc_def_var(ncid, "wmo_1st_t", NC_FLOAT, 3, &dims[0], &wmo1tid));
00145      NC(nc_def_var(ncid, "wmo_1st_q", NC_FLOAT, 3, &dims[0], &wmo1qid));
00146      NC(nc_def_var(ncid, "wmo_2nd_z", NC_FLOAT, 3, &dims[0], &wmo2zid));
00147      NC(nc_def_var(ncid, "wmo_2nd_p", NC_FLOAT, 3, &dims[0], &wmo2pid));
00148      NC(nc_def_var(ncid, "wmo_2nd_t", NC_FLOAT, 3, &dims[0], &wmo2tid));
00149      NC(nc_def_var(ncid, "wmo_2nd_q", NC_FLOAT, 3, &dims[0], &wmo2qid));
00150
00151      /* Set attributes... */
00152      add_text_attribute(ncid, "time", "units", "s");
00153      add_text_attribute(ncid, "time", "long_name",
00154          "seconds since 2000-01-01, 00:00 UTC");
00155      add_text_attribute(ncid, "lon", "units", "degrees_east");
00156      add_text_attribute(ncid, "lon", "long_name", "longitude");
00157      add_text_attribute(ncid, "lat", "units", "degrees_north");
00158      add_text_attribute(ncid, "lat", "long_name", "latitude");
00159
00160      add_text_attribute(ncid, "clp_z", "units", "km");
00161      add_text_attribute(ncid, "clp_z", "long_name", "cold point height");
00162      add_text_attribute(ncid, "clp_p", "units", "hPa");
00163      add_text_attribute(ncid, "clp_p", "long_name", "cold point pressure");
00164      add_text_attribute(ncid, "clp_t", "units", "K");
00165      add_text_attribute(ncid, "clp_t", "long_name",
00166          "cold point temperature");
00167      add_text_attribute(ncid, "clp_q", "units", "ppv");
00168      add_text_attribute(ncid, "clp_q", "long_name",
00169          "cold point water vapor");
00170
00171      add_text_attribute(ncid, "dyn_z", "units", "km");
00172      add_text_attribute(ncid, "dyn_z", "long_name",
00173          "dynamical tropopause height");
00174      add_text_attribute(ncid, "dyn_p", "units", "hPa");
00175      add_text_attribute(ncid, "dyn_p", "long_name",
00176          "dynamical tropopause pressure");
00177      add_text_attribute(ncid, "dyn_t", "units", "K");
00178      add_text_attribute(ncid, "dyn_t", "long_name",
00179          "dynamical tropopause temperature");
00180      add_text_attribute(ncid, "dyn_q", "units", "ppv");
00181      add_text_attribute(ncid, "dyn_q", "long_name",
00182          "dynamical tropopause water vapor");
00183
00184      add_text_attribute(ncid, "wmo_1st_z", "units", "km");
00185      add_text_attribute(ncid, "wmo_1st_z", "long_name",
00186          "WMO 1st tropopause height");
00187      add_text_attribute(ncid, "wmo_1st_p", "units", "hPa");
00188      add_text_attribute(ncid, "wmo_1st_p", "long_name",
00189          "WMO 1st tropopause pressure");
00190      add_text_attribute(ncid, "wmo_1st_t", "units", "K");
00191      add_text_attribute(ncid, "wmo_1st_t", "long_name",
00192          "WMO 1st tropopause temperature");
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      add_text_attribute(ncid, "wmo_2nd_z", "units", "km");
00198      add_text_attribute(ncid, "wmo_2nd_z", "long_name",
00199          "WMO 2nd tropopause height");
00200      add_text_attribute(ncid, "wmo_2nd_p", "units", "hPa");
00201      add_text_attribute(ncid, "wmo_2nd_p", "long_name",
00202          "WMO 2nd tropopause pressure");
00203      add_text_attribute(ncid, "wmo_2nd_t", "units", "K");
00204      add_text_attribute(ncid, "wmo_2nd_t", "long_name",
00205          "WMO 2nd tropopause temperature");
00206      add_text_attribute(ncid, "wmo_2nd_q", "units", "ppv");

```

```

00207     add_text_attribute(ncid, "wmo_2nd_q", "long_name",
00208                       "WMO 2nd tropopause water vapor");
00209
00210     /* End definition... */
00211     NC(nc_enddef(ncid));
00212
00213     /* Write longitude and latitude... */
00214     NC(nc_put_var_double(ncid, latid, lats));
00215     NC(nc_put_var_double(ncid, lonid, lons));
00216 }
00217
00218 /* Write time... */
00219 start[0] = (size_t) nt;
00220 count[0] = 1;
00221 start[1] = 0;
00222 count[1] = (size_t) ny;
00223 start[2] = 0;
00224 count[2] = (size_t) nx;
00225 NC(nc_put_vara_double(ncid, timid, start, count, &met->time));
00226
00227 /* Get cold point... */
00228 ctl.met_tropo = 2;
00229 read_met_tropo(&ctl, met);
00230 #pragma omp parallel for default(shared) private(ix,iy)
00231 for (ix = 0; ix < nx; ix++)
00232     for (iy = 0; iy < ny; iy++) {
00233         intpol_met_space(met, 100, lons[ix], lats[iy], NULL,
00234                         &pt[iy * nx + ix], NULL, NULL, NULL,
00235                         NULL, NULL, NULL, NULL, NULL);
00236         intpol_met_space(met, pt[iy * nx + ix], lons[ix], lats[iy],
00237                         NULL, NULL, &zt[iy * nx + ix], &tt[iy * nx + ix],
00238                         NULL, NULL, NULL, NULL, &qt[iy * nx + ix], NULL);
00239     }
00240
00241 /* Write data... */
00242 NC(nc_put_vara_double(ncid, clpzid, start, count, zt));
00243 NC(nc_put_vara_double(ncid, clppid, start, count, pt));
00244 NC(nc_put_vara_double(ncid, clptid, start, count, tt));
00245 NC(nc_put_vara_double(ncid, clpqid, start, count, qt));
00246
00247 /* Get dynamical tropopause... */
00248 ctl.met_tropo = 5;
00249 read_met_tropo(&ctl, met);
00250 #pragma omp parallel for default(shared) private(ix,iy)
00251 for (ix = 0; ix < nx; ix++)
00252     for (iy = 0; iy < ny; iy++) {
00253         intpol_met_space(met, 100, lons[ix], lats[iy], NULL,
00254                         &pt[iy * nx + ix], NULL, NULL, NULL,
00255                         NULL, NULL, NULL, NULL, NULL);
00256         intpol_met_space(met, pt[iy * nx + ix], lons[ix], lats[iy],
00257                         NULL, NULL, &zt[iy * nx + ix], &tt[iy * nx + ix],
00258                         NULL, NULL, NULL, NULL, &qt[iy * nx + ix], NULL);
00259     }
00260
00261 /* Write data... */
00262 NC(nc_put_vara_double(ncid, dynzid, start, count, zt));
00263 NC(nc_put_vara_double(ncid, dynpid, start, count, pt));
00264 NC(nc_put_vara_double(ncid, dyntid, start, count, tt));
00265 NC(nc_put_vara_double(ncid, dynqid, start, count, qt));
00266
00267 /* Get WMO 1st tropopause... */
00268 ctl.met_tropo = 3;
00269 read_met_tropo(&ctl, met);
00270 #pragma omp parallel for default(shared) private(ix,iy)
00271 for (ix = 0; ix < nx; ix++)
00272     for (iy = 0; iy < ny; iy++) {
00273         intpol_met_space(met, 100, lons[ix], lats[iy], NULL,
00274                         &pt[iy * nx + ix], NULL, NULL, NULL,
00275                         NULL, NULL, NULL, NULL, NULL);
00276         intpol_met_space(met, pt[iy * nx + ix], lons[ix], lats[iy],
00277                         NULL, NULL, &zt[iy * nx + ix], &tt[iy * nx + ix],
00278                         NULL, NULL, NULL, NULL, &qt[iy * nx + ix], NULL);
00279     }
00280
00281 /* Write data... */
00282 NC(nc_put_vara_double(ncid, wmolzid, start, count, zt));
00283 NC(nc_put_vara_double(ncid, wmolpid, start, count, pt));
00284 NC(nc_put_vara_double(ncid, wmoltid, start, count, tt));
00285 NC(nc_put_vara_double(ncid, wmolqid, start, count, qt));
00286
00287 /* Get WMO 2nd tropopause... */
00288 ctl.met_tropo = 4;
00289 read_met_tropo(&ctl, met);
00290 #pragma omp parallel for default(shared) private(ix,iy)
00291 for (ix = 0; ix < nx; ix++)
00292     for (iy = 0; iy < ny; iy++) {
00293         intpol_met_space(met, 100, lons[ix], lats[iy], NULL,

```

```

00294             &pt[iy * nx + ix], NULL, NULL, NULL,
00295             NULL, NULL, NULL, NULL, NULL);
00296     intpol_met_space(met, pt[iy * nx + ix], lons[ix], lats[iy],
00297                     NULL, NULL, &zt[iy * nx + ix], &tt[iy * nx + ix],
00298                     NULL, NULL, NULL, NULL, &qt[iy * nx + ix], NULL);
00299 }
00300
00301 /* Write data... */
00302 NC(nc_put_vara_double(ncid, wmo2zid, start, count, zt));
00303 NC(nc_put_vara_double(ncid, wmo2pid, start, count, pt));
00304 NC(nc_put_vara_double(ncid, wmo2tid, start, count, tt));
00305 NC(nc_put_vara_double(ncid, wmo2qid, start, count, qt));
00306
00307 /* Increment time step counter... */
00308 nt++;
00309 }
00310
00311 /* Close file... */
00312 NC(nc_close(ncid));
00313
00314 /* Free... */
00315 free(met);
00316
00317 return EXIT_SUCCESS;
00318 }
00319
00320 /*****
00321
00322 void add_text_attribute(
00323     int ncid,
00324     char *varname,
00325     char *attrname,
00326     char *text) {
00327
00328     int varid;
00329
00330     NC(nc_inq_varid(ncid, varname, &varid));
00331     NC(nc_put_att_text(ncid, varid, attrname, strlen(text), text));
00332 }

```

Index

add_text_attribute
 tropo.c, 242

atm_basename
 ctl_t, 17

atm_conv.c, 26
 main, 26

atm_dist.c, 28
 main, 28

atm_dt_out
 ctl_t, 17

atm_filter
 ctl_t, 17

atm_gpfile
 ctl_t, 17

atm_init.c, 36
 main, 37

atm_split.c, 40
 main, 40

atm_stat.c, 45
 main, 45

atm_stride
 ctl_t, 17

atm_t, 3
 lat, 4
 lon, 4
 np, 3
 p, 3
 q, 4
 time, 3

atm_type
 ctl_t, 17

balloon
 ctl_t, 15

cache_t, 4
 iso_n, 5
 iso_ps, 5
 iso_ts, 5
 iso_var, 5
 tsig, 5
 up, 5
 usig, 6
 vp, 5
 vsig, 6
 wp, 5
 wsig, 6

cart2geo
 libtrac.c, 60
 libtrac.h, 141

clim_hno3
 libtrac.c, 60
 libtrac.h, 141

clim_tropo
 libtrac.c, 61
 libtrac.h, 141

csi_basename
 ctl_t, 17

csi_dt_out
 ctl_t, 18

csi_lat0
 ctl_t, 19

csi_lat1
 ctl_t, 19

csi_lon0
 ctl_t, 18

csi_lon1
 ctl_t, 19

csi_modmin
 ctl_t, 18

csi_nx
 ctl_t, 18

csi_ny
 ctl_t, 19

csi_nz
 ctl_t, 18

csi_obsfile
 ctl_t, 18

csi_obsmin
 ctl_t, 18

csi_z0
 ctl_t, 18

csi_z1
 ctl_t, 18

ctl_t, 6
 atm_basename, 17
 atm_dt_out, 17
 atm_filter, 17
 atm_gpfile, 17
 atm_stride, 17
 atm_type, 17
 balloon, 15
 csi_basename, 17
 csi_dt_out, 18
 csi_lat0, 19
 csi_lat1, 19
 csi_lon0, 18
 csi_lon1, 19
 csi_modmin, 18
 csi_nx, 18
 csi_ny, 19
 csi_nz, 18
 csi_obsfile, 18
 csi_obsmin, 18
 csi_z0, 18
 csi_z1, 18
 direction, 13
 dt_met, 14
 dt_mod, 14
 ens_basename, 22
 grid_basename, 19

[grid_dt_out](#), 19
[grid_gpfile](#), 19
[grid_lat0](#), 20
[grid_lat1](#), 20
[grid_lon0](#), 20
[grid_lon1](#), 20
[grid_nx](#), 20
[grid_ny](#), 20
[grid_nz](#), 19
[grid_sparse](#), 19
[grid_z0](#), 20
[grid_z1](#), 20
[isosurf](#), 15
[met_dp](#), 14
[met_dt_out](#), 15
[met_dx](#), 14
[met_dy](#), 14
[met_geopot](#), 15
[met_np](#), 15
[met_p](#), 15
[met_sp](#), 15
[met_stage](#), 15
[met_sx](#), 14
[met_sy](#), 14
[met_tropo](#), 15
[molmass](#), 16
[nq](#), 11
[prof_basename](#), 20
[prof_lat0](#), 21
[prof_lat1](#), 22
[prof_lon0](#), 21
[prof_lon1](#), 21
[prof_nx](#), 21
[prof_ny](#), 21
[prof_nz](#), 21
[prof_obsfile](#), 21
[prof_z0](#), 21
[prof_z1](#), 21
[psc_h2o](#), 17
[psc_hno3](#), 17
[qnt_ens](#), 11
[qnt_format](#), 11
[qnt_h2o](#), 12
[qnt_m](#), 11
[qnt_name](#), 11
[qnt_o3](#), 12
[qnt_p](#), 12
[qnt_ps](#), 11
[qnt_pt](#), 12
[qnt_pv](#), 13
[qnt_r](#), 11
[qnt_rho](#), 11
[qnt_stat](#), 13
[qnt_t](#), 12
[qnt_theta](#), 13
[qnt_tice](#), 13
[qnt_tnat](#), 13
[qnt_tsts](#), 13
[qnt_u](#), 12
[qnt_unit](#), 11
[qnt_v](#), 12
[qnt_vh](#), 13
[qnt_vz](#), 13
[qnt_w](#), 12
[qnt_z](#), 12
[stat_basename](#), 22
[stat_lat](#), 22
[stat_lon](#), 22
[stat_r](#), 22
[t_start](#), 14
[t_stop](#), 14
[tdec_strat](#), 16
[tdec_trop](#), 16
[turb_dx_strat](#), 16
[turb_dx_trop](#), 16
[turb_dz_strat](#), 16
[turb_dz_trop](#), 16
[turb_mesox](#), 16
[turb_mesoz](#), 16

[day2doy](#)
 [libtrac.c](#), 61
 [libtrac.h](#), 142
[day2doy.c](#), 51
 [main](#), 51
[direction](#)
 [ctl_t](#), 13
[doy2day](#)
 [libtrac.c](#), 62
 [libtrac.h](#), 142
[doy2day.c](#), 52
 [main](#), 53
[dt_met](#)
 [ctl_t](#), 14
[dt_mod](#)
 [ctl_t](#), 14

[ens_basename](#)
 [ctl_t](#), 22
[extract.c](#), 54
 [main](#), 54

[geo2cart](#)
 [libtrac.c](#), 62
 [libtrac.h](#), 143
[get_met](#)
 [libtrac.c](#), 62
 [libtrac.h](#), 143
[get_met_help](#)
 [libtrac.c](#), 64
 [libtrac.h](#), 145
[get_met_replace](#)
 [libtrac.c](#), 65
 [libtrac.h](#), 145
[grid_basename](#)
 [ctl_t](#), 19
[grid_dt_out](#)

- ctl_t, 19
- grid_gpfile
 - ctl_t, 19
- grid_lat0
 - ctl_t, 20
- grid_lat1
 - ctl_t, 20
- grid_lon0
 - ctl_t, 20
- grid_lon1
 - ctl_t, 20
- grid_nx
 - ctl_t, 20
- grid_ny
 - ctl_t, 20
- grid_nz
 - ctl_t, 19
- grid_sparse
 - ctl_t, 19
- grid_z0
 - ctl_t, 20
- grid_z1
 - ctl_t, 20
- h2o
 - met_t, 25
- intpol_met_2d
 - libtrac.c, 65
 - libtrac.h, 145
- intpol_met_3d
 - libtrac.c, 65
 - libtrac.h, 146
- intpol_met_space
 - libtrac.c, 66
 - libtrac.h, 146
- intpol_met_time
 - libtrac.c, 67
 - libtrac.h, 147
- iso_n
 - cache_t, 5
- iso_ps
 - cache_t, 5
- iso_ts
 - cache_t, 5
- iso_var
 - cache_t, 5
- isosurf
 - ctl_t, 15
- jsec2time
 - libtrac.c, 68
 - libtrac.h, 148
- jsec2time.c, 56
 - main, 57
- lat
 - atm_t, 4
 - met_t, 24
- libtrac.c, 58
 - cart2geo, 60
 - clim_hno3, 60
 - clim_tropo, 61
 - day2doy, 61
 - doy2day, 62
 - geo2cart, 62
 - get_met, 62
 - get_met_help, 64
 - get_met_replace, 65
 - intpol_met_2d, 65
 - intpol_met_3d, 65
 - intpol_met_space, 66
 - intpol_met_time, 67
 - jsec2time, 68
 - locate_irr, 68
 - locate_reg, 69
 - read_atm, 69
 - read_ctl, 71
 - read_met, 75
 - read_met_extrapolate, 77
 - read_met_geopot, 78
 - read_met_help, 80
 - read_met_ml2pl, 81
 - read_met_periodic, 81
 - read_met_pv, 82
 - read_met_sample, 83
 - read_met_tropo, 85
 - scan_ctl, 87
 - spline, 88
 - stddev, 88
 - time2jsec, 88
 - timer, 89
 - write_atm, 89
 - write_csi, 91
 - write_ens, 93
 - write_grid, 95
 - write_prof, 97
 - write_station, 100
- libtrac.h, 139
 - cart2geo, 141
 - clim_hno3, 141
 - clim_tropo, 141
 - day2doy, 142
 - doy2day, 142
 - geo2cart, 143
 - get_met, 143
 - get_met_help, 145
 - get_met_replace, 145
 - intpol_met_2d, 145
 - intpol_met_3d, 146
 - intpol_met_space, 146
 - intpol_met_time, 147
 - jsec2time, 148
 - locate_irr, 149
 - locate_reg, 149
 - read_atm, 149
 - read_ctl, 152

- read_met, 155
- read_met_extrapolate, 158
- read_met_geopot, 158
- read_met_help, 161
- read_met_ml2pl, 161
- read_met_periodic, 162
- read_met_pv, 162
- read_met_sample, 164
- read_met_tropo, 165
- scan_ctl, 167
- spline, 168
- stddev, 169
- time2jsec, 169
- timer, 169
- write_atm, 170
- write_csi, 172
- write_ens, 174
- write_grid, 176
- write_prof, 178
- write_station, 180
- locate_irr
 - libtrac.c, 68
 - libtrac.h, 149
- locate_reg
 - libtrac.c, 69
 - libtrac.h, 149
- lon
 - atm_t, 4
 - met_t, 24
- main
 - atm_conv.c, 26
 - atm_dist.c, 28
 - atm_init.c, 37
 - atm_split.c, 40
 - atm_stat.c, 45
 - day2doy.c, 51
 - doy2day.c, 53
 - extract.c, 54
 - jsec2time.c, 57
 - met_map.c, 191
 - met_prof.c, 196
 - met_sample.c, 201
 - met_zm.c, 204
 - time2jsec.c, 210
 - trac.c, 224
 - tropo.c, 242
- met_dp
 - ctl_t, 14
- met_dt_out
 - ctl_t, 15
- met_dx
 - ctl_t, 14
- met_dy
 - ctl_t, 14
- met_geopot
 - ctl_t, 15
- met_map.c, 191
 - main, 191
- met_np
 - ctl_t, 15
- met_p
 - ctl_t, 15
- met_prof.c, 195
 - main, 196
- met_sample.c, 200
 - main, 201
- met_sp
 - ctl_t, 15
- met_stage
 - ctl_t, 15
- met_sx
 - ctl_t, 14
- met_sy
 - ctl_t, 14
- met_t, 22
 - h2o, 25
 - lat, 24
 - lon, 24
 - np, 24
 - nx, 23
 - ny, 24
 - o3, 25
 - p, 24
 - pl, 25
 - ps, 24
 - pt, 24
 - pv, 25
 - t, 24
 - time, 23
 - u, 25
 - v, 25
 - w, 25
 - z, 24
- met_tropo
 - ctl_t, 15
- met_zm.c, 204
 - main, 204
- module_advection
 - trac.c, 212
- module_decay
 - trac.c, 212
- module_diffusion_init
 - trac.c, 213
- module_diffusion_meso
 - trac.c, 214
- module_diffusion_rng
 - trac.c, 215
- module_diffusion_turb
 - trac.c, 216
- module_isosurf
 - trac.c, 218
- module_isosurf_init
 - trac.c, 217
- module_meteo
 - trac.c, 219
- module_position

trac.c, [220](#)
module_sedi
trac.c, [221](#)
molmass
ctl_t, [16](#)

np
atm_t, [3](#)
met_t, [24](#)

nq
ctl_t, [11](#)

nx
met_t, [23](#)

ny
met_t, [24](#)

o3
met_t, [25](#)

p
atm_t, [3](#)
met_t, [24](#)

pl
met_t, [25](#)
prof_basename
ctl_t, [20](#)
prof_lat0
ctl_t, [21](#)
prof_lat1
ctl_t, [22](#)
prof_lon0
ctl_t, [21](#)
prof_lon1
ctl_t, [21](#)
prof_nx
ctl_t, [21](#)
prof_ny
ctl_t, [21](#)
prof_nz
ctl_t, [21](#)
prof_obsfile
ctl_t, [21](#)
prof_z0
ctl_t, [21](#)
prof_z1
ctl_t, [21](#)

ps
met_t, [24](#)
psc_h2o
ctl_t, [17](#)
psc_hno3
ctl_t, [17](#)

pt
met_t, [24](#)

pv
met_t, [25](#)

q
atm_t, [4](#)

qnt_ens
ctl_t, [11](#)
qnt_format
ctl_t, [11](#)
qnt_h2o
ctl_t, [12](#)
qnt_m
ctl_t, [11](#)
qnt_name
ctl_t, [11](#)
qnt_o3
ctl_t, [12](#)
qnt_p
ctl_t, [12](#)
qnt_ps
ctl_t, [11](#)
qnt_pt
ctl_t, [12](#)
qnt_pv
ctl_t, [13](#)
qnt_r
ctl_t, [11](#)
qnt_rho
ctl_t, [11](#)
qnt_stat
ctl_t, [13](#)
qnt_t
ctl_t, [12](#)
qnt_theta
ctl_t, [13](#)
qnt_tice
ctl_t, [13](#)
qnt_tnat
ctl_t, [13](#)
qnt_tsts
ctl_t, [13](#)
qnt_u
ctl_t, [12](#)
qnt_unit
ctl_t, [11](#)
qnt_v
ctl_t, [12](#)
qnt_vh
ctl_t, [13](#)
qnt_vz
ctl_t, [13](#)
qnt_w
ctl_t, [12](#)
qnt_z
ctl_t, [12](#)

read_atm
libtrac.c, [69](#)
libtrac.h, [149](#)
read_ctl
libtrac.c, [71](#)
libtrac.h, [152](#)
read_met
libtrac.c, [75](#)

- libtrac.h, 155
- read_met_extrapolate
 - libtrac.c, 77
 - libtrac.h, 158
- read_met_geopot
 - libtrac.c, 78
 - libtrac.h, 158
- read_met_help
 - libtrac.c, 80
 - libtrac.h, 161
- read_met_ml2pl
 - libtrac.c, 81
 - libtrac.h, 161
- read_met_periodic
 - libtrac.c, 81
 - libtrac.h, 162
- read_met_pv
 - libtrac.c, 82
 - libtrac.h, 162
- read_met_sample
 - libtrac.c, 83
 - libtrac.h, 164
- read_met_tropo
 - libtrac.c, 85
 - libtrac.h, 165
- rng
 - trac.c, 228
- scan_ctl
 - libtrac.c, 87
 - libtrac.h, 167
- spline
 - libtrac.c, 88
 - libtrac.h, 168
- stat_basename
 - ctl_t, 22
- stat_lat
 - ctl_t, 22
- stat_lon
 - ctl_t, 22
- stat_r
 - ctl_t, 22
- stddev
 - libtrac.c, 88
 - libtrac.h, 169
- t
 - met_t, 24
- t_start
 - ctl_t, 14
- t_stop
 - ctl_t, 14
- tdec_strat
 - ctl_t, 16
- tdec_trop
 - ctl_t, 16
- time
 - atm_t, 3
 - met_t, 23
- time2jsec
 - libtrac.c, 88
 - libtrac.h, 169
- time2jsec.c, 209
 - main, 210
- timer
 - libtrac.c, 89
 - libtrac.h, 169
- trac.c, 211
 - main, 224
 - module_advection, 212
 - module_decay, 212
 - module_diffusion_init, 213
 - module_diffusion_meso, 214
 - module_diffusion_rng, 215
 - module_diffusion_turb, 216
 - module_isosurf, 218
 - module_isosurf_init, 217
 - module_meteo, 219
 - module_position, 220
 - module_sedi, 221
 - rng, 228
 - write_output, 222
- tropo.c, 242
 - add_text_attribute, 242
 - main, 242
- tsig
 - cache_t, 5
- turb_dx_strat
 - ctl_t, 16
- turb_dx_trop
 - ctl_t, 16
- turb_dz_strat
 - ctl_t, 16
- turb_dz_trop
 - ctl_t, 16
- turb_mesox
 - ctl_t, 16
- turb_mesoz
 - ctl_t, 16
- u
 - met_t, 25
- up
 - cache_t, 5
- usig
 - cache_t, 6
- v
 - met_t, 25
- vp
 - cache_t, 5
- vsig
 - cache_t, 6
- w
 - met_t, 25
- wp
 - cache_t, 5

- write_atm
 - libtrac.c, [89](#)
 - libtrac.h, [170](#)
- write_csi
 - libtrac.c, [91](#)
 - libtrac.h, [172](#)
- write_ens
 - libtrac.c, [93](#)
 - libtrac.h, [174](#)
- write_grid
 - libtrac.c, [95](#)
 - libtrac.h, [176](#)
- write_output
 - trac.c, [222](#)
- write_prof
 - libtrac.c, [97](#)
 - libtrac.h, [178](#)
- write_station
 - libtrac.c, [100](#)
 - libtrac.h, [180](#)
- wsig
 - cache_t, [6](#)
- z
 - met_t, [24](#)