MPTRAC

Generated by Doxygen 1.8.11

ii CONTENTS

Contents

1	Main	Page		2	
2	Data Structure Index				
	2.1	Data S	tructures	2	
3	File	Index		2	
	3.1	File Lis	st	2	
4	4 Data Structure Documentation		ure Documentation	3	
	4.1	atm_t S	Struct Reference	3	
		4.1.1	Detailed Description	4	
		4.1.2	Field Documentation	4	
	4.2	ctl_t St	ruct Reference	6	
		4.2.1	Detailed Description	10	
		4.2.2	Field Documentation	10	
	4.3	met_t s	Struct Reference	22	
		4.3.1	Detailed Description	23	
		4.3.2	Field Documentation	23	
5	File	Docume	entation	25	
	5.1	atm_cc	onv.c File Reference	25	
		5.1.1	Detailed Description	25	
		5.1.2	Function Documentation	26	
	5.2	atm_cc	onv.c	26	
	5.3	atm_di	st.c File Reference	27	
		5.3.1	Detailed Description	27	
		5.3.2	Function Documentation	27	
	5.4	atm_di	st.c	31	
	5.5	atm_in	it.c File Reference	34	
		5.5.1	Detailed Description	34	
		5.5.2	Function Documentation	35	

5.6	atm_init.c	36
5.7	atm_split.c File Reference	38
	5.7.1 Detailed Description	38
	5.7.2 Function Documentation	38
5.8	atm_split.c	40
5.9	atm_stat.c File Reference	42
	5.9.1 Detailed Description	42
	5.9.2 Function Documentation	42
5.10	atm_stat.c	44
5.11	day2doy.c File Reference	46
	5.11.1 Detailed Description	46
	5.11.2 Function Documentation	46
5.12	day2doy.c	47
5.13	doy2day.c File Reference	47
	5.13.1 Detailed Description	47
	5.13.2 Function Documentation	48
5.14	doy2day.c	48
5.15	extract.c File Reference	49
	5.15.1 Detailed Description	49
	5.15.2 Function Documentation	49
5.16	extract.c	50
5.17	jsec2time.c File Reference	51
	5.17.1 Detailed Description	52
	5.17.2 Function Documentation	52
5.18	jsec2time.c	52
5.19	libtrac.c File Reference	53
	5.19.1 Detailed Description	54
	5.19.2 Function Documentation	55
5.20	libtrac.c	98
5.21	libtrac.h File Reference	133

	5.21.1 Detailed Description	135
	5.21.2 Function Documentation	135
5.22	libtrac.h	178
5.23	met_map.c File Reference	186
	5.23.1 Detailed Description	186
	5.23.2 Function Documentation	186
5.24	met_map.c	189
5.25	met_prof.c File Reference	190
	5.25.1 Detailed Description	190
	5.25.2 Function Documentation	191
5.26	met_prof.c	193
5.27	met_sample.c File Reference	194
	5.27.1 Detailed Description	195
	5.27.2 Function Documentation	195
5.28	met_sample.c	196
5.29	met_zm.c File Reference	198
	5.29.1 Detailed Description	198
	5.29.2 Function Documentation	199
5.30	met_zm.c	200
5.31	smago.c File Reference	202
	5.31.1 Detailed Description	202
	5.31.2 Function Documentation	202
5.32	smago.c	204
5.33	time2jsec.c File Reference	206
	5.33.1 Detailed Description	206
	5.33.2 Function Documentation	206
5.34	time2jsec.c	207
5.35	trac.c File Reference	207
	5.35.1 Detailed Description	208
	5.35.2 Function Documentation	208
5.36	trac.c	222
5.37	wind.c File Reference	234
	5.37.1 Detailed Description	234
	5.37.2 Function Documentation	234
5.38	wind.c	236

Index 241

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
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 atmospheric data files	25	
atm_dist.c		
Calculate transport deviations of trajectories	27	
atm_init.c		
Create atmospheric data file with initial air parcel positions	34	
atm_split.c		
Split air parcels into a larger number of parcels	38	
atm_stat.c		
Calculate air parcel statistics	42	
day2doy.c		
Convert date to day of year	46	

doy2day.c	
Convert day of year to date	47
extract.c Extract single trajectory from atmospheric data files	49
jsec2time.c Convert Julian seconds to date	51
libtrac.c MPTRAC library definitions	53
libtrac.h MPTRAC library declarations	133
met_map.c Extract global map from meteorological data	186
met_prof.c Extract vertical profile from meteorological data	190
met_sample.c Sample meteorological data at given geolocations	194
met_zm.c Extract zonal mean from meteorological data	198
smago.c Estimate horizontal diffusivity based on Smagorinsky theory	202
time2jsec.c Convert date to Julian seconds	206
trac.c Lagrangian particle dispersion model	207
wind.c Create meteorological data files with synthetic wind fields	234

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).
    • 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 cache_time [EX][EY][EP]
           Cache for reference time of wind standard deviations.

    float cache_usig [EX][EY][EP]

           Cache for zonal wind standard deviations.

    float cache_vsig [EX][EY][EP]

           Cache for meridional wind standard deviations.

    float cache_wsig [EX][EY][EP]

           Cache for vertical velocity standard deviations.
4.1.1 Detailed Description
Atmospheric data.
Definition at line 592 of file libtrac.h.
4.1.2 Field Documentation
4.1.2.1 int atm_t::np
Number of air pacels.
Definition at line 595 of file libtrac.h.
4.1.2.2 double atm_t::time[NP]
Time [s].
Definition at line 598 of file libtrac.h.
4.1.2.3 double atm_t::p[NP]
Pressure [hPa].
Definition at line 601 of file libtrac.h.
```

```
4.1.2.4 double atm_t::lon[NP]
Longitude [deg].
Definition at line 604 of file libtrac.h.
4.1.2.5 double atm_t::lat[NP]
Latitude [deg].
Definition at line 607 of file libtrac.h.
4.1.2.6 double atm_t::q[NQ][NP]
Quantity data (for various, user-defined attributes).
Definition at line 610 of file libtrac.h.
4.1.2.7 float atm_t::up[NP]
Zonal wind perturbation [m/s].
Definition at line 613 of file libtrac.h.
4.1.2.8 float atm_t::vp[NP]
Meridional wind perturbation [m/s].
Definition at line 616 of file libtrac.h.
4.1.2.9 float atm_t::wp[NP]
Vertical velocity perturbation [hPa/s].
Definition at line 619 of file libtrac.h.
4.1.2.10 double atm_t::cache_time[EX][EY][EP]
Cache for reference time of wind standard deviations.
Definition at line 622 of file libtrac.h.
4.1.2.11 float atm_t::cache_usig[EX][EY][EP]
Cache for zonal wind standard deviations.
Definition at line 625 of file libtrac.h.
4.1.2.12 float atm_t::cache_vsig[EX][EY][EP]
Cache for meridional wind standard deviations.
Definition at line 628 of file libtrac.h.
```

```
4.1.2.13 float atm_t::cache_wsig[EX][EY][EP]
```

Cache for vertical velocity standard deviations.

Definition at line 631 of file libtrac.h.

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

· libtrac.h

4.2 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) $[m^{\wedge}2/s]$.

double turb_dz_trop

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

• double turb_dz_strat

Vertical turbulent diffusion coefficient (stratosphere) $[m^{\wedge}2/s]$.

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]

Basename of atmospheric data files.

char atm_gpfile [LEN]

Gnuplot file for atmospheric data.

· double atm_dt_out

Time step for atmospheric data output [s].

· int atm_filter

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

int atm_type

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

char csi basename [LEN]

Basename of CSI data files.

double csi_dt_out

Time step for CSI data output [s].

char csi_obsfile [LEN]

Observation data file for CSI analysis.

· double csi_obsmin

Minimum observation index to trigger detection.

• double csi_modmin

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

• int csi_nz

Number of altitudes of gridded CSI data.

double csi_z0

Lower altitude of gridded CSI data [km].

double csi_z1

Upper altitude of gridded CSI data [km].

int csi_nx

Number of longitudes of gridded CSI data.

• double csi_lon0

Lower longitude of gridded CSI data [deg].

double csi_lon1

Upper longitude of gridded CSI data [deg].

• int csi_ny

Number of latitudes of gridded CSI data.

• double csi_lat0

Lower latitude of gridded CSI data [deg].

double csi_lat1

Upper latitude of gridded CSI data [deg].

char grid_basename [LEN]

Basename of grid data files.

• char grid_gpfile [LEN]

Gnuplot file for gridded data.

double grid_dt_out

Time step for gridded data output [s].

· int grid_sparse

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

• int grid_nz

Number of altitudes of gridded data.

• double grid_z0

Lower altitude of gridded data [km].

• double grid_z1

Upper altitude of gridded data [km].

• int grid_nx

Number of longitudes of gridded data.

double grid_lon0

Lower longitude of gridded data [deg].

double grid_lon1

Upper longitude of gridded data [deg].

int grid_ny

Number of latitudes of gridded data.

• double grid_lat0

Lower latitude of gridded data [deg].

double grid_lat1

Upper latitude of gridded data [deg].

char prof_basename [LEN]

Basename for profile output file.

• char prof_obsfile [LEN]

Observation data file for profile output.

• int prof_nz

Number of altitudes of gridded profile data.

double prof z0

Lower altitude of gridded profile data [km].

double prof_z1

Upper altitude of gridded profile data [km].

• int prof_nx

Number of longitudes of gridded profile data.

• double prof_lon0

Lower longitude of gridded profile data [deg].

double prof_lon1

Upper longitude of gridded profile data [deg].

int prof_ny

Number of latitudes of gridded profile data.

double prof_lat0

Lower latitude of gridded profile data [deg].

double prof lat1

Upper latitude of gridded profile data [deg].

• char ens_basename [LEN]

Basename of ensemble data file.

char stat_basename [LEN]

Basename of station data file.

double stat_lon

Longitude of station [deg].

double stat_lat

Latitude of station [deg].

• double stat_r

Search radius around station [km].

4.2.1 Detailed Description

Control parameters.

Definition at line 273 of file libtrac.h.

4.2.2 Field Documentation

4.2.2.1 int ctl_t::nq

Number of quantities.

Definition at line 276 of file libtrac.h.

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

Quantity names.

Definition at line 279 of file libtrac.h.

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

Quantity units.

Definition at line 282 of file libtrac.h.

4.2.2.4 char ctl_t::qnt_format[NQ][LEN] Quantity output format. Definition at line 285 of file libtrac.h. 4.2.2.5 int ctl_t::qnt_ens Quantity array index for ensemble IDs. Definition at line 288 of file libtrac.h. 4.2.2.6 int ctl_t::qnt_m Quantity array index for mass. Definition at line 291 of file libtrac.h. 4.2.2.7 int ctl_t::qnt_rho Quantity array index for particle density. Definition at line 294 of file libtrac.h. 4.2.2.8 int ctl_t::qnt_r Quantity array index for particle radius. Definition at line 297 of file libtrac.h. 4.2.2.9 int ctl_t::qnt_ps Quantity array index for surface pressure. Definition at line 300 of file libtrac.h. 4.2.2.10 int ctl_t::qnt_pt Quantity array index for tropopause pressure. Definition at line 303 of file libtrac.h. 4.2.2.11 int ctl_t::qnt_z Quantity array index for geopotential height. Definition at line 306 of file libtrac.h. 4.2.2.12 int ctl_t::qnt_p

Quantity array index for pressure.

Definition at line 309 of file libtrac.h.

```
4.2.2.13 int ctl_t::qnt_t
Quantity array index for temperature.
Definition at line 312 of file libtrac.h.
4.2.2.14 int ctl_t::qnt_u
Quantity array index for zonal wind.
Definition at line 315 of file libtrac.h.
4.2.2.15 int ctl_t::qnt_v
Quantity array index for meridional wind.
Definition at line 318 of file libtrac.h.
4.2.2.16 int ctl_t::qnt_w
Quantity array index for vertical velocity.
Definition at line 321 of file libtrac.h.
4.2.2.17 int ctl_t::qnt_h2o
Quantity array index for water vapor vmr.
Definition at line 324 of file libtrac.h.
4.2.2.18 int ctl_t::qnt_o3
Quantity array index for ozone vmr.
Definition at line 327 of file libtrac.h.
4.2.2.19 int ctl_t::qnt_theta
Quantity array index for potential temperature.
Definition at line 330 of file libtrac.h.
4.2.2.20 int ctl_t::qnt_vh
Quantity array index for horizontal wind.
Definition at line 333 of file libtrac.h.
4.2.2.21 int ctl_t::qnt_vz
Quantity array index for vertical velocity.
Definition at line 336 of file libtrac.h.
```

```
4.2.2.22 int ctl_t::qnt_pv
Quantity array index for potential vorticity.
Definition at line 339 of file libtrac.h.
4.2.2.23 int ctl_t::qnt_tice
Quantity array index for T_ice.
Definition at line 342 of file libtrac.h.
4.2.2.24 int ctl_t::qnt_tsts
Quantity array index for T_STS.
Definition at line 345 of file libtrac.h.
4.2.2.25 int ctl_t::qnt_tnat
Quantity array index for T_NAT.
Definition at line 348 of file libtrac.h.
4.2.2.26 int ctl_t::qnt_stat
Quantity array index for station flag.
Definition at line 351 of file libtrac.h.
4.2.2.27 int ctl_t::direction
Direction flag (1=forward calculation, -1=backward calculation).
Definition at line 354 of file libtrac.h.
4.2.2.28 double ctl_t::t_start
Start time of simulation [s].
Definition at line 357 of file libtrac.h.
4.2.2.29 double ctl_t::t_stop
Stop time of simulation [s].
Definition at line 360 of file libtrac.h.
4.2.2.30 double ctl_t::dt_mod
Time step of simulation [s].
Definition at line 363 of file libtrac.h.
```

```
4.2.2.31 double ctl_t::dt_met
Time step of meteorological data [s].
Definition at line 366 of file libtrac.h.
4.2.2.32 int ctl_t::met_dx
Stride for longitudes.
Definition at line 369 of file libtrac.h.
4.2.2.33 int ctl_t::met_dy
Stride for latitudes.
Definition at line 372 of file libtrac.h.
4.2.2.34 int ctl_t::met_dp
Stride for pressure levels.
Definition at line 375 of file libtrac.h.
4.2.2.35 int ctl_t::met_sx
Smoothing for longitudes.
Definition at line 378 of file libtrac.h.
4.2.2.36 int ctl_t::met_sy
Smoothing for latitudes.
Definition at line 381 of file libtrac.h.
4.2.2.37 int ctl_t::met_sp
Smoothing for pressure levels.
Definition at line 384 of file libtrac.h.
4.2.2.38 int ctl_t::met_np
Number of target pressure levels.
Definition at line 387 of file libtrac.h.
4.2.2.39 double ctl_t::met_p[EP]
Target pressure levels [hPa].
Definition at line 390 of file libtrac.h.
```

```
4.2.2.40 int ctl_t::met_tropo
Tropopause definition (0=none, 1=clim, 2=cold point, 3=WMO 1st, 4=WMO 2nd).
Definition at line 394 of file libtrac.h.
4.2.2.41 char ctl_t::met_geopot[LEN]
Surface geopotential data file.
Definition at line 397 of file libtrac.h.
4.2.2.42 double ctl_t::met_dt_out
Time step for sampling of meteo data along trajectories [s].
Definition at line 400 of file libtrac.h.
4.2.2.43 char ctl_t::met_stage[LEN]
Command to stage meteo data.
Definition at line 403 of file libtrac.h.
4.2.2.44 int ctl_t::isosurf
Isosurface parameter (0=none, 1=pressure, 2=density, 3=theta, 4=balloon).
Definition at line 407 of file libtrac.h.
4.2.2.45 char ctl_t::balloon[LEN]
Balloon position filename.
Definition at line 410 of file libtrac.h.
4.2.2.46 double ctl_t::turb_dx_trop
Horizontal turbulent diffusion coefficient (troposphere) [m^2/s].
Definition at line 413 of file libtrac.h.
4.2.2.47 double ctl_t::turb_dx_strat
Horizontal turbulent diffusion coefficient (stratosphere) [m<sup>2</sup>/s].
Definition at line 416 of file libtrac.h.
4.2.2.48 double ctl_t::turb_dz_trop
Vertical turbulent diffusion coefficient (troposphere) [m<sup>2</sup>/s].
Definition at line 419 of file libtrac.h.
```

```
4.2.2.49 double ctl_t::turb_dz_strat
Vertical turbulent diffusion coefficient (stratosphere) [m<sup>2</sup>/s].
Definition at line 422 of file libtrac.h.
4.2.2.50 double ctl_t::turb_mesox
Horizontal scaling factor for mesoscale wind fluctuations.
Definition at line 425 of file libtrac.h.
4.2.2.51 double ctl_t::turb_mesoz
Vertical scaling factor for mesoscale wind fluctuations.
Definition at line 428 of file libtrac.h.
4.2.2.52 double ctl_t::molmass
Molar mass [g/mol].
Definition at line 431 of file libtrac.h.
4.2.2.53 double ctl_t::tdec_trop
Life time of particles (troposphere) [s].
Definition at line 434 of file libtrac.h.
4.2.2.54 double ctl_t::tdec_strat
Life time of particles (stratosphere) [s].
Definition at line 437 of file libtrac.h.
4.2.2.55 double ctl_t::psc_h2o
H2O volume mixing ratio for PSC analysis.
Definition at line 440 of file libtrac.h.
4.2.2.56 double ctl_t::psc_hno3
HNO3 volume mixing ratio for PSC analysis.
Definition at line 443 of file libtrac.h.
4.2.2.57 char ctl_t::atm_basename[LEN]
Basename of atmospheric data files.
Definition at line 446 of file libtrac.h.
```

```
4.2.2.58 char ctl_t::atm_gpfile[LEN]
Gnuplot file for atmospheric data.
Definition at line 449 of file libtrac.h.
4.2.2.59 double ctl_t::atm_dt_out
Time step for atmospheric data output [s].
Definition at line 452 of file libtrac.h.
4.2.2.60 int ctl_t::atm_filter
Time filter for atmospheric data output (0=no, 1=yes).
Definition at line 455 of file libtrac.h.
4.2.2.61 int ctl_t::atm_type
Type of atmospheric data files (0=ASCII, 1=binary, 2=netCDF).
Definition at line 458 of file libtrac.h.
4.2.2.62 char ctl_t::csi_basename[LEN]
Basename of CSI data files.
Definition at line 461 of file libtrac.h.
4.2.2.63 double ctl_t::csi_dt_out
Time step for CSI data output [s].
Definition at line 464 of file libtrac.h.
4.2.2.64 char ctl_t::csi_obsfile[LEN]
Observation data file for CSI analysis.
Definition at line 467 of file libtrac.h.
4.2.2.65 double ctl_t::csi_obsmin
Minimum observation index to trigger detection.
Definition at line 470 of file libtrac.h.
4.2.2.66 double ctl_t::csi_modmin
Minimum column density to trigger detection [kg/m^{\wedge}2].
Definition at line 473 of file libtrac.h.
```

```
4.2.2.67 int ctl_t::csi_nz
Number of altitudes of gridded CSI data.
Definition at line 476 of file libtrac.h.
4.2.2.68 double ctl_t::csi_z0
Lower altitude of gridded CSI data [km].
Definition at line 479 of file libtrac.h.
4.2.2.69 double ctl_t::csi_z1
Upper altitude of gridded CSI data [km].
Definition at line 482 of file libtrac.h.
4.2.2.70 int ctl_t::csi_nx
Number of longitudes of gridded CSI data.
Definition at line 485 of file libtrac.h.
4.2.2.71 double ctl_t::csi_lon0
Lower longitude of gridded CSI data [deg].
Definition at line 488 of file libtrac.h.
4.2.2.72 double ctl_t::csi_lon1
Upper longitude of gridded CSI data [deg].
Definition at line 491 of file libtrac.h.
4.2.2.73 int ctl_t::csi_ny
Number of latitudes of gridded CSI data.
Definition at line 494 of file libtrac.h.
4.2.2.74 double ctl_t::csi_lat0
Lower latitude of gridded CSI data [deg].
Definition at line 497 of file libtrac.h.
4.2.2.75 double ctl_t::csi_lat1
Upper latitude of gridded CSI data [deg].
```

Definition at line 500 of file libtrac.h.

```
4.2.2.76 char ctl_t::grid_basename[LEN]
Basename of grid data files.
Definition at line 503 of file libtrac.h.
4.2.2.77 char ctl_t::grid_gpfile[LEN]
Gnuplot file for gridded data.
Definition at line 506 of file libtrac.h.
4.2.2.78 double ctl_t::grid_dt_out
Time step for gridded data output [s].
Definition at line 509 of file libtrac.h.
4.2.2.79 int ctl_t::grid_sparse
Sparse output in grid data files (0=no, 1=yes).
Definition at line 512 of file libtrac.h.
4.2.2.80 int ctl_t::grid_nz
Number of altitudes of gridded data.
Definition at line 515 of file libtrac.h.
4.2.2.81 double ctl_t::grid_z0
Lower altitude of gridded data [km].
Definition at line 518 of file libtrac.h.
4.2.2.82 double ctl_t::grid_z1
Upper altitude of gridded data [km].
Definition at line 521 of file libtrac.h.
4.2.2.83 int ctl_t::grid_nx
Number of longitudes of gridded data.
Definition at line 524 of file libtrac.h.
4.2.2.84 double ctl_t::grid_lon0
Lower longitude of gridded data [deg].
Definition at line 527 of file libtrac.h.
```

```
4.2.2.85 double ctl_t::grid_lon1
Upper longitude of gridded data [deg].
Definition at line 530 of file libtrac.h.
4.2.2.86 int ctl_t::grid_ny
Number of latitudes of gridded data.
Definition at line 533 of file libtrac.h.
4.2.2.87 double ctl_t::grid_lat0
Lower latitude of gridded data [deg].
Definition at line 536 of file libtrac.h.
4.2.2.88 double ctl_t::grid_lat1
Upper latitude of gridded data [deg].
Definition at line 539 of file libtrac.h.
4.2.2.89 char ctl_t::prof_basename[LEN]
Basename for profile output file.
Definition at line 542 of file libtrac.h.
4.2.2.90 char ctl_t::prof_obsfile[LEN]
Observation data file for profile output.
Definition at line 545 of file libtrac.h.
4.2.2.91 int ctl_t::prof_nz
Number of altitudes of gridded profile data.
Definition at line 548 of file libtrac.h.
4.2.2.92 double ctl_t::prof_z0
Lower altitude of gridded profile data [km].
Definition at line 551 of file libtrac.h.
4.2.2.93 double ctl_t::prof_z1
Upper altitude of gridded profile data [km].
Definition at line 554 of file libtrac.h.
```

4.2.2.94 int ctl_t::prof_nx Number of longitudes of gridded profile data. Definition at line 557 of file libtrac.h. 4.2.2.95 double ctl_t::prof_lon0 Lower longitude of gridded profile data [deg]. Definition at line 560 of file libtrac.h. 4.2.2.96 double ctl_t::prof_lon1 Upper longitude of gridded profile data [deg]. Definition at line 563 of file libtrac.h. 4.2.2.97 int ctl_t::prof_ny Number of latitudes of gridded profile data. Definition at line 566 of file libtrac.h. 4.2.2.98 double ctl_t::prof_lat0 Lower latitude of gridded profile data [deg]. Definition at line 569 of file libtrac.h. 4.2.2.99 double ctl_t::prof_lat1 Upper latitude of gridded profile data [deg]. Definition at line 572 of file libtrac.h. 4.2.2.100 char ctl_t::ens_basename[LEN] Basename of ensemble data file. Definition at line 575 of file libtrac.h. 4.2.2.101 char ctl_t::stat_basename[LEN] Basename of station data file. Definition at line 578 of file libtrac.h. 4.2.2.102 double ctl_t::stat_lon Longitude of station [deg].

Definition at line 581 of file libtrac.h.

```
4.2.2.103 double ctl_t::stat_lat
Latitude of station [deg].
Definition at line 584 of file libtrac.h.
4.2.2.104 double ctl_t::stat_r
Search radius around station [km].
Definition at line 587 of file libtrac.h.
The documentation for this struct was generated from the following file:
    · libtrac.h
     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].
4.3.1 Detailed Description
Meteorological data.
Definition at line 636 of file libtrac.h.
4.3.2 Field Documentation
4.3.2.1 double met_t::time
Time [s].
Definition at line 639 of file libtrac.h.
4.3.2.2 int met_t::nx
Number of longitudes.
Definition at line 642 of file libtrac.h.
4.3.2.3 int met_t::ny
Number of latitudes.
Definition at line 645 of file libtrac.h.
4.3.2.4 int met_t::np
Number of pressure levels.
Definition at line 648 of file libtrac.h.
4.3.2.5 double met_t::lon[EX]
Longitude [deg].
Definition at line 651 of file libtrac.h.
```

```
4.3.2.6 double met_t::lat[EY]
Latitude [deg].
Definition at line 654 of file libtrac.h.
4.3.2.7 double met_t::p[EP]
Pressure [hPa].
Definition at line 657 of file libtrac.h.
4.3.2.8 double met_t::ps[EX][EY]
Surface pressure [hPa].
Definition at line 660 of file libtrac.h.
4.3.2.9 double met_t::pt[EX][EY]
Tropopause pressure [hPa].
Definition at line 663 of file libtrac.h.
4.3.2.10 float met_t::z[EX][EY][EP]
Geopotential height [km].
Definition at line 666 of file libtrac.h.
4.3.2.11 float met_t::t[EX][EY][EP]
Temperature [K].
Definition at line 669 of file libtrac.h.
4.3.2.12 float met_t::u[EX][EY][EP]
Zonal wind [m/s].
Definition at line 672 of file libtrac.h.
4.3.2.13 float met_t::v[EX][EY][EP]
Meridional wind [m/s].
Definition at line 675 of file libtrac.h.
4.3.2.14 float met_t::w[EX][EY][EP]
Vertical wind [hPa/s].
Definition at line 678 of file libtrac.h.
```

5 File Documentation 25

```
4.3.2.15 float met_t::pv[EX][EY][EP]
Potential vorticity [PVU].
Definition at line 681 of file libtrac.h.
4.3.2.16 float met_t::h2o[EX][EY][EP]
Water vapor volume mixing ratio [1].
Definition at line 684 of file libtrac.h.
4.3.2.17 float met_t::o3[EX][EY][EP]
Ozone volume mixing ratio [1].
Definition at line 687 of file libtrac.h.
4.3.2.18 float met_t::pl[EX][EY][EP]
Pressure on model levels [hPa].
Definition at line 690 of file libtrac.h.
The documentation for this struct was generated from the following file:
    · libtrac.h
    File Documentation
5.1 atm_conv.c File Reference
Convert file format of atmospheric data files.
Functions
    • int main (int argc, char *argv[])
5.1.1 Detailed Description
Convert file format of atmospheric data files.
Definition in file atm_conv.c.
```

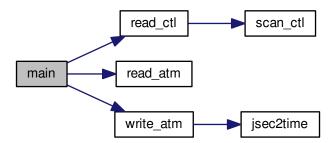
5.1.2 Function Documentation

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

Definition at line 27 of file atm_conv.c.

```
00029
00030
00031
00032
00033
       atm_t *atm;
00034
00035
       /* Check arguments... */
00036
       if (argc < 6)
         00037
00038
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
       read_atm(argv[2], &ctl, atm);
00049
00050
       /* Write atmospheric data... */
       ctl.atm_type = atoi(argv[5]);
write_atm(argv[4], &ctl, atm, 0);
00051
00052
00053
00054
       /* Free... */
       free(atm);
00056
00057
       return EXIT_SUCCESS;
00058 }
```

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
```

```
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 <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>.
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... */
ctl.atm_type = atoi(argv[3]);
00047
00048
        read_atm(argv[2], &ctl, atm);
00049
00050
        /\star Write atmospheric data..
00051
        ctl.atm_type = atoi(argv[5]);
00052
        write_atm(argv[4], &ctl, atm, 0);
00054
         /* Free... */
00055
        free(atm);
00056
00057
        return EXIT_SUCCESS;
00058 }
```

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.

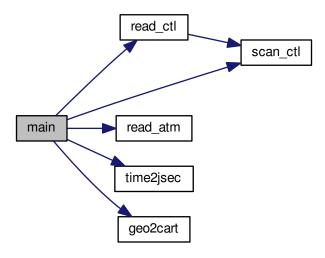
```
00030
00031
          ctl_t ctl;
00032
00033
          atm t *atm1, *atm2;
00034
          FILE *out;
00036
00037
          char tstr[LEN];
00038
00039
          double ahtd, agtd[NQ], atce1[NQ], atce2[NQ], avtd, lat0, lat1,
            *lat1_old, *lat2_old, *lh1, *lh2, lon0, lon1, *lon1_old, *lon2_old, *lv1, *lv2, p0, p1, *q1, *q2, rhtd, rqtd[NQ], rtce1[NQ], rtce2[NQ], rvtd, t, t0 = 0, x0[3], x1[3], x2[3], z1, *z1_old, z2, *z2_old;
00040
00041
00042
00043
00044
          int ens, f, ip, iq, np, year, mon, day, hour, min;
00045
00046
           /* Allocate... */
          ALLOC(atm1, atm_t, 1);
ALLOC(atm2, atm_t, 1);
00047
00048
00049
          ALLOC(lon1_old, double,
00050
                 NP);
          ALLOC(lat1_old, double,
00051
00052
                 NP);
00053
          ALLOC(z1_old, double,
                  NP);
00054
00055
          ALLOC(lh1, double,
00056
                 NP);
          ALLOC(lv1, double,
00057
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(1h2, double,
00066
                 NP);
          ALLOC(1v2, double,
00067
00068
                  NP);
00069
          ALLOC(q1, double,
00070
                 NQ * NP);
00071
          ALLOC(q2, double,
                  NQ * NP);
00072
00073
00074
          /* Check arguments... */
00075
          if (argc < 5)</pre>
            00076
00077
00078
          /* Read control parameters... */
00080
          read_ctl(argv[1], argc, argv, &ctl);
          read_ct1(argv[1], argc, argv, &ct1);
ens = (int) scan_ct1(argv[1], argc, argv, "DIST_ENS", -1, "-1", NULL);
p0 = P(scan_ct1(argv[1], argc, argv, "DIST_Z0", -1, "-1000", NULL));
p1 = P(scan_ct1(argv[1], argc, argv, "DIST_Z1", -1, "1000", NULL));
lat0 = scan_ct1(argv[1], argc, argv, "DIST_LATO", -1, "-1000", NULL);
lat1 = scan_ct1(argv[1], argc, argv, "DIST_LAT1", -1, "1000", NULL);
lon0 = scan_ct1(argv[1], argc, argv, "DIST_LON0", -1, "-1000", NULL);
lon1 = scan_ct1(argv[1], argc, argv, "DIST_LON0", -1, "1000", NULL);
00081
00082
00083
00084
00086
00087
00088
          /* Write info... */
00089
00090
          printf("Write transport deviations: %s\n", argv[2]);
00091
00092
          /* Create output file... */
if (!(out = fopen(argv[2], "w")))
00093
            ERRMSG("Cannot create file!");
00094
00095
00096
          /* Write header... */
00097
          fprintf(out,
                     "# $1 = time [s] \n"
00098
00099
                     "# $2 = trajectory time [s]\n"
                     "# $3 = AHTD [km] \n"
00100
                     "# $4 = RHTD [%%]\n" "# $5 = AVTD [km]\n" "# $6 = RVTD [%%]\n");
00101
           for (iq = 0; iq < ctl.nq; iq++)</pre>
00102
00103
            fprintf(out,
                        "# $%d = AQTD (%s) [%s]\n"
00104
00105
                        "# \$%d = RQTD (%s) [%%]\n",
                        7 + 2 * iq, ctl.qnt_name[iq], ctl.qnt_unit[iq],
8 + 2 * iq, ctl.qnt_name[iq]);
00106
00107
00108
          for (iq = 0; iq < ctl.nq; iq++)</pre>
00109
            fprintf(out,
                        "# $%d = ATCE_1 (%s) [%s]\n"
00110
00111
                        "# \$%d = RTCE_1 (%s) [%%]\n",
                       7 + 2 * ctl.nq + 2 * iq, ctl.qnt_name[iq], ctl.qnt_unit[iq], 8 + 2 * ctl.nq + 2 * iq, ctl.qnt_name[iq]);
00112
00113
          for (iq = 0; iq < ctl.nq; iq++)</pre>
00114
00115
            fprintf(out,
```

```
00116
                   "# $%d = ATCE_2 (%s) [%s]\n"
                   "# \$%d = RTCE_2 (%s) [%%]\n",
00117
                   7 + 4 * ctl.nq + 2 * iq, ctl.qnt_name[iq], ctl.qnt_unit[iq], 8 + 4 * ctl.nq + 2 * iq, ctl.qnt_name[iq]);
00118
00119
        fprintf(out, "# \$%d = number of particlesn\n", 7 + 6 * ctl.nq);
00120
00121
00122
         /* Loop over file pairs... */
00123
        for (f = 3; f < argc; f += 2) {</pre>
00124
00125
           /* Read atmopheric data... */
          read_atm(argv[f], &ctl, atm1);
read_atm(argv[f + 1], &ctl, atm2);
00126
00127
00128
00129
           /* Check if structs match... */
00130
           if (atm1->np != atm2->np)
00131
            ERRMSG("Different numbers of parcels!");
           for (ip = 0; ip < atml->np; ip++)
  if (gsl_finite(atml->time[ip]) && gsl_finite(atm2->time[ip])
        && atml->time[ip] != atm2->time[ip])
00132
00133
00134
00135
               ERRMSG("Times do not match!");
00136
00137
           /* Get time from filename... */
           00138
           vear = atoi(tstr);
00139
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00140
00141
           mon = atoi(tstr);
00142
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00143
           day = atoi(tstr);
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00144
00145
          hour = atoi(tstr);
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00146
00147
           min = atoi(tstr);
00148
          time2jsec(year, mon, day, hour, min, 0, 0, &t);
00149
00150
           /* Save initial data... */
          if (f == 3) {
00151
            t0 = t;
00152
             for (iq = 0; iq < ctl.nq; iq++)
00153
              for (ip = 0; ip < atml->np; ip++) {
    q1[iq * NP + ip] = atml->q[iq][ip];
00154
00155
00156
                 q2[iq * NP + ip] = atm2->q[iq][ip];
               }
00157
00158
          }
00159
           /* Init... */
00160
00161
           np = 0;
00162
           ahtd = avtd = rhtd = rvtd = 0;
00163
           for (iq = 0; iq < ctl.nq; iq++)</pre>
             aqtd[iq] = atcel[iq] = atce2[iq] = rqtd[iq] = rtce1[iq] = rtce2[iq] = 0;
00164
00165
00166
           /* Loop over air parcels... */
00167
          for (ip = 0; ip < atm1->np; ip++) {
00168
00169
             /* Check data... */
             if (!gsl_finite(atm1->time[ip]) || !gsl_finite(atm2->time[ip]))
00170
00171
              continue;
00172
00173
             /* Check ensemble ID... */
00174
             if (ens >= 0 && ctl.qnt_ens >= 0 && atml->q[ctl.qnt_ens][ip] != ens)
00175
               continue;
             if (ens \geq= 0 && ctl.qnt_ens \geq= 0 && atm2-\geqq[ctl.qnt_ens][ip] != ens)
00176
00177
              continue;
00178
00179
             /* Check spatial range... */
00180
             if (atm1->p[ip] > p0 || atm1->p[ip] < p1
                 || atm1->lon[ip] < lon0 || atm1->lon[ip] > lon1
00181
                 || atm1->lat[ip] < lat0 || atm1->lat[ip] > lat1)
00182
00183
               continue:
00184
             if (atm2->p[ip] > p0 || atm2->p[ip] < p1</pre>
                 00185
00186
00187
               continue;
00188
             /* Convert coordinates... */
geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00189
00190
00191
00192
             z1 = Z(atm1->p[ip]);
00193
             z2 = Z(atm2->p[ip]);
00194
00195
             /* Calculate absolute transport deviations... */
             ahtd += DIST(x1, x2);
avtd += fabs(z1 - z2);
00196
00197
             for (iq = 0; iq < ctl.nq; iq++)</pre>
00198
00199
               aqtd[iq] += fabs(atm1->q[iq][ip] - atm2->q[iq][ip]);
00200
00201
             /\star Calculate relative transport deviations... \star/
             if (f > 3) {
00202
```

```
00203
00204
                 /* Get trajectory lengths... */
                 geo2cart(0, lon1_old[ip], lat1_old[ip], x0);
lh1[ip] += DIST(x0, x1);
00205
00206
                 lv1[ip] += fabs(z1_old[ip] - z1);
00207
00208
                 geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
lh2[ip] += DIST(x0, x2);
00209
00210
00211
                 lv2[ip] += fabs(z2_old[ip] - z2);
00212
00213
                 /* Get relative transport deviations... */
                 if (lh1[ip] + lh2[ip] > 0)
00214
                   rhtd += 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00215
00216
                 if (lv1[ip] + lv2[ip] > 0)
00217
                    rvtd += 200. * fabs(z1 - z2) / (lv1[ip] + lv2[ip]);
                 for (iq = 0; iq < ctl.nq; iq++)
  rqtd[iq] += 200. * fabs(atml->q[iq][ip] - atm2->q[iq][ip])
00218
00219
00220
                      / (fabs(atml->q[iq][ip]) + fabs(atm2->q[iq][ip]));
00222
                 /* Get tracer conservation errors... */
00223
                 for (iq = 0; iq < ctl.nq; iq++)</pre>
                   atcel[iq] += fabs(atml->q[iq][ip] - ql[iq * NP + ip]);
rtcel[iq] += 200. * fabs(atml->q[iq][ip] - ql[iq * NP + ip])
/ (fabs(atml->q[iq][ip]) + fabs(ql[iq * NP + ip]));
00224
00225
00226
                   / (tabs(atml->q[iq][ip]) + Tabs(q[iq * NF + ip])),
atce2[iq] += fabs(atm2->q[iq][ip] - q2[iq * NF + ip]);
rtce2[iq] += 200. * fabs(atm2->q[iq][ip] - q2[iq * NF + ip])
00227
00228
00229
                       / (fabs(atm2->q[iq][ip]) + fabs(q2[iq \star NP + ip]));
00230
00231
00232
00233
               /\star Save positions of air parcels... \star/
               lon1_old[ip] = atm1->lon[ip];
lat1_old[ip] = atm1->lat[ip];
00234
00235
00236
               z1\_old[ip] = z1;
00237
              lon2_old[ip] = atm2->lon[ip];
00238
00239
               lat2_old[ip] = atm2->lat[ip];
              z2_old[ip] = z2;
00240
00241
00242
               /* Increment air parcel counter... */
00243
              np++;
            }
00244
00245
            00246
00247
00248
            for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");</pre>
00249
00250
               fprintf(out, ctl.qnt_format[iq], aqtd[iq] / np);
fprintf(out, " ");
00251
00252
              fprintf(out, ctl.qnt_format[iq], rqtd[iq] / np);
00253
00254
            for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl.qnt_format[iq], atcel[iq] / np);
  fprintf(out, " ");</pre>
00255
00256
00257
00258
               fprintf(out, ctl.qnt_format[iq], rtcel[iq] / np);
00259
00260
            for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");</pre>
00261
00262
               fprintf(out, ctl.qnt_format[iq], atce2[iq] / np);
fprintf(out, " ");
00263
00264
00265
               fprintf(out, ctl.qnt_format[iq], rtce2[iq] / np);
00266
00267
            fprintf(out, " %d\n", np);
00268
00269
00270
          /* Close file... */
00271
         fclose(out);
00272
00273
          /* Free... */
00274
         free(atm1);
00275
         free(atm2);
00276
         free(lon1_old);
00277
         free(lat1 old);
00278
         free(z1_old);
00279
         free(lh1);
00280
         free(lv1);
00281
         free(lon2_old);
00282
         free(lat2 old):
00283
         free(z2 old);
00284
         free(lh2);
00285
         free(lv2);
00286
00287
         return EXIT_SUCCESS;
00288 }
```

5.4 atm dist.c 31

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 it under the terms of the GNU General Public License as published by
00005
          the Free Software Foundation, either version 3 of the License, or
00006
00007
          (at your option) any later version.
00008
          MPTRAC is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
00009
00010
00011
          MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
          GNU General Public License for more details.
00013
          You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
00016
          Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00017
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028
          int argc,
          char *argv[]) {
00029
00030
00031
          ctl_t ctl;
00032
00033
          atm_t *atm1, *atm2;
00034
00035
          FILE *out;
00036
00037
          char tstr[LEN];
00038
          double ahtd, aqtd[NQ], atcel[NQ], atcel[NQ], avtd, lat0, lat1,
00039
           *lat1_old, *lat2_old, *lh1, *lh2, lon0, lon1, *lon1_old, *lon2_old, *lv1, *lv2, p0, p1, *q1, *q2, rhtd, rqtd[NQ], rtce1[NQ], rtce2[NQ], rvtd, t, t0 = 0, x0[3], x1[3], x2[3], z1, *z1_old, z2, *z2_old;
00040
00041
00042
00043
00044
          int ens, f, ip, iq, np, year, mon, day, hour, min;
00045
00046
           /* Allocate... */
          ALLOC(atm1, atm_t, 1);
ALLOC(atm2, atm_t, 1);
00047
00048
          ALLOC(lon1_old, double,
```

```
00050
                NP);
00051
         ALLOC(lat1_old, double,
00052
                NP);
         ALLOC(z1_old, double,
00053
00054
                NP);
         ALLOC(lh1, double,
00055
00056
                 NP);
00057
         ALLOC(lv1, double,
00058
                 NP);
         ALLOC(lon2_old, double,
00059
00060
                NP);
00061
         ALLOC(lat2 old, double,
00062
                 NP);
00063
         ALLOC(z2_old, double,
00064
                NP);
00065
         ALLOC(1h2, double,
00066
                NP);
00067
         ALLOC(1v2, double,
00068
                NP);
         ALLOC(q1, double,
00069
00070
                NQ * NP);
00071
         ALLOC(q2, double,
00072
                 NQ * NP);
00073
00074
         /* Check arguments... */
00075
         if (argc < 5)
00076
            ERRMSG("Give parameters: <ctl> <outfile> <atmla> <atmlb>"
00077
                     " [<atm2a> <atm2b> ...]");
00078
00079
         /* Read control parameters... */
         /* Read control parameters... */
read_ctl(argv[1], argc, argv, &ctl);
ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-1", NULL);
p0 = P(scan_ctl(argv[1], argc, argv, "DIST_20", -1, "-1000", NULL));
p1 = P(scan_ctl(argv[1], argc, argv, "DIST_21", -1, "1000", NULL));
lat0 = scan_ctl(argv[1], argc, argv, "DIST_LAT0", -1, "-1000", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "DIST_LAT1", -1, "1000", NULL);
lon0 = scan_ctl(argv[1], argc, argv, "DIST_LON0", -1, "-1000", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "DIST_LON0", -1, "-1000", NULL);
00080
00081
00082
00083
00084
00085
00086
00088
00089
          /* Write info... */
00090
         printf("Write transport deviations: sn", argv[2]);
00091
         /* Create output file... */
if (!(out = fopen(argv[2], "w")))
00092
00093
            ERRMSG("Cannot create file!");
00094
00095
00096
          /* Write header... */
         00097
00098
                    "# $2 = trajectory time [s]\n"
00099
                   "# $3 = AHTD [km]\n"
"# $4 = RHTD [%%]\n" "# $5 = AVTD [km]\n" "# $6 = RVTD [%%]\n");
00100
00101
00102
          for (iq = 0; iq < ctl.nq; iq++)</pre>
          00103
00104
                      "# $$d = RQTD (%s) [%\]\n",
7 + 2 * iq, ctl.qnt_name[iq], ctl.qnt_unit[iq],
00105
00106
00107
                      8 + 2 * iq, ctl.qnt_name[iq]);
00108
         for (iq = 0; iq < ctl.nq; iq++)</pre>
           00109
00110
                      "# \$%d = RTCE_1 (%s) [%%]\n",
00111
                      7 + 2 * ctl.nq + 2 * iq, ctl.qnt_name[iq], ctl.qnt_unit[iq], 8 + 2 * ctl.nq + 2 * iq, ctl.qnt_name[iq]);
00112
00113
00114
         for (iq = 0; iq < ctl.nq; iq++)</pre>
          fprintf(out,
00115
                       "# $%d = ATCE 2 (%s) [%s]\n"
00116
                      "# $%d = RTCE_2 (%s) [%%]\n",
00117
                      7 + 4 * ctl.nq + 2 * iq, ctl.qnt_name[iq], ctl.qnt_unit[iq],
00118
                      8 + 4 * ctl.nq + 2 * iq, ctl.qnt_name[iq]);
00119
00120
         fprintf(out, "# \$d = number of particlesn, 7 + 6 * ctl.nq);
00121
         /* Loop over file pairs... */
for (f = 3; f < argc; f += 2) {
00122
00123
00124
00125
            /* Read atmopheric data...
00126
            read_atm(argv[f], &ctl, atml);
00127
            read_atm(argv[f + 1], &ctl, atm2);
00128
00129
            /* Check if structs match... */
            if (atm1->np != atm2->np)
00130
              ERRMSG("Different numbers of parcels!");
00131
00132
            for (ip = 0; ip < atml->np; ip++)
00133
               if (gsl_finite(atm1->time[ip]) && gsl_finite(atm2->time[ip])
00134
                    && atm1->time[ip] != atm2->time[ip])
                 ERRMSG("Times do not match!");
00135
00136
```

5.4 atm dist.c 33

```
/* Get time from filename... */
           sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00138
00139
           year = atoi(tstr);
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00140
00141
           mon = atoi(tstr);
00142
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
           day = atoi(tstr);
00143
00144
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00145
           hour = atoi(tstr);
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00146
           min = atoi(tstr);
00147
00148
           time2jsec(year, mon, day, hour, min, 0, 0, &t);
00149
00150
           /* Save initial data... */
00151
           if (f == 3) {
00152
             t0 = t;
             for (iq = 0; iq < ctl.nq; iq++)
  for (ip = 0; ip < atml->np; ip++) {
    ql[iq * NP + ip] = atml->q[iq][ip];
00153
00154
00155
                 q2[iq * NP + ip] = atm2->q[iq][ip];
00156
00157
00158
           }
00159
           /* Init... */
00160
00161
           np = 0;
           ahtd = avtd = rhtd = rvtd = 0;
00162
00163
           for (iq = 0; iq < ctl.nq; iq++)</pre>
00164
             aqtd[iq] = atce1[iq] = atce2[iq] = rqtd[iq] = rtce1[iq] = rtce2[iq] = 0;
00165
00166
           /* Loop over air parcels... */
00167
           for (ip = 0; ip < atm1->np; ip++) {
00168
00169
              /* Check data... */
00170
              if (!gsl_finite(atm1->time[ip]) || !gsl_finite(atm2->time[ip]))
00171
00172
             /* Check ensemble ID... */
if (ens >= 0 && ctl.qnt_ens >= 0 && atml->q[ctl.qnt_ens][ip] != ens)
00173
00175
                continue;
00176
              if (ens >= 0 && ctl.qnt_ens >= 0 && atm2->q[ctl.qnt_ens][ip] != ens)
00177
                continue;
00178
00179
              /* Check spatial range... */
             00180
00181
00182
                  || atm1->lat[ip] < lat0 || atm1->lat[ip] > lat1)
00183
                continue;
             if (atm2->p[ip] > p0 || atm2->p[ip] < p1
    || atm2->lon[ip] < lon0 || atm2->lon[ip] > lon1
    || atm2->lat[ip] < lat0 || atm2->lat[ip] > lat1)
00184
00185
00186
00187
                continue;
00188
              /* Convert coordinates... */
00189
             geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00190
00191
              z1 = Z(atm1->p[ip]);
00192
             z2 = Z(atm2->p[ip]);
00194
00195
              /\star Calculate absolute transport deviations... \star/
00196
              ahtd += DIST(x1, x2);
              avtd += fabs(z1 - z2);
00197
              for (iq = 0; iq < ctl.nq; iq++)</pre>
00198
00199
                aqtd[iq] += fabs(atm1->q[iq][ip] - atm2->q[iq][ip]);
00200
00201
              /* Calculate relative transport deviations... */
00202
              if (f > 3) {
00203
                /* Get trajectory lengths... */
geo2cart(0, lon1_old[ip], lat1_old[ip], x0);
lh1[ip] += DIST(x0, x1);
00204
00205
00207
                lv1[ip] += fabs(z1_old[ip] - z1);
00208
                geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
lh2[ip] += DIST(x0, x2);
lv2[ip] += fabs(z2_old[ip] - z2);
00209
00210
00211
00212
00213
                /* Get relative transport deviations... */
00214
                if (lh1[ip] + lh2[ip] > 0)
00215
                  rhtd += 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00216
                if (lv1[ip] + lv2[ip] > 0)
                  rvtd += 200. * fabs(z1 - z2) / (lv1[ip] + lv2[ip]);
00217
                for (iq = 0; iq < ctl.nq; iq++)</pre>
00218
00219
                 rqtd[iq] += 200. * fabs(atm1->q[iq][ip] - atm2->q[iq][ip])
00220
                     / (fabs(atm1->q[iq][ip]) + fabs(atm2->q[iq][ip]));
00221
00222
                /\star Get tracer conservation errors... \star/
00223
                for (iq = 0; iq < ctl.nq; iq++) {</pre>
```

```
atcel[iq] += fabs(atml->q[iq][ip] - ql[iq * NP + ip]);
                    atcer[iq] += labs(atml->q[iq][ip] - q1[iq * NP + ip]);
rtce1[iq] += 200. * fabs(atml->q[iq][ip] - q1[iq * NP + ip])
    / (fabs(atml->q[iq][ip]) + fabs(q1[iq * NP + ip]));
atce2[iq] += fabs(atm2->q[iq][ip] - q2[iq * NP + ip]);
rtce2[iq] += 200. * fabs(atm2->q[iq][ip] - q2[iq * NP + ip])
    / (fabs(atm2->q[iq][ip]) + fabs(q2[iq * NP + ip]));
00226
00227
00228
00229
00231
00232
00233
                /\star Save positions of air parcels... \star/
               lon1_old[ip] = atm1->lon[ip];
lat1_old[ip] = atm1->lat[ip];
00234
00235
00236
               z1_old[ip] = z1;
00237
               lon2_old[ip] = atm2->lon[ip];
lat2_old[ip] = atm2->lat[ip];
z2_old[ip] = z2;
00238
00239
00240
00241
00242
                /* Increment air parcel counter... */
00243
               np++;
00244
00245
            00246
00247
00248
00250
               fprintf(out, ctl.qnt_format[iq], aqtd[iq] / np);
fprintf(out, " ");
00251
00252
00253
                fprintf(out, ctl.qnt_format[iq], rqtd[iq] / np);
00254
             for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");</pre>
00255
00256
                fprintf(out, ctl.qnt_format[iq], atcel[iq] / np);
fprintf(out, " ");
00257
00258
                fprintf(out, ctl.qnt_format[iq], rtcel[iq] / np);
00259
00260
            for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");</pre>
00261
00262
                fprintf(out, ctl.qnt_format[iq], atce2[iq] / np);
fprintf(out, " ");
00263
00264
                fprintf(out, ctl.qnt_format[iq], rtce2[iq] / np);
00265
00266
00267
             fprintf(out, " %d\n", np);
00268
00269
00270
          /* Close file... */
00271
          fclose(out);
00272
          /* Free... */
00273
          free(atm1);
00275
          free(atm2);
00276
          free(lon1_old);
00277
          free(lat1_old);
00278
          free (z1_old);
00279
          free(lh1);
00280
          free(lv1);
00281
          free(lon2_old);
00282
          free(lat2_old);
00283
          free(z2_old);
00284
          free(1h2):
00285
          free(lv2);
00286
          return EXIT_SUCCESS;
00288 }
```

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

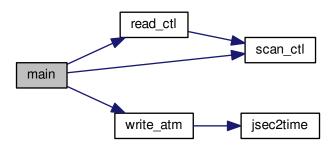
5.5.2.1 int main (int argc, char * argv[])

Definition at line 27 of file atm init.c.

```
00029
00030
00031
                atm_t *atm;
00032
00033
               ctl t ctl;
00034
               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... */
               /* Read control parameters... */
read_ctl(argv[1], argc, argv, &ctl);
t0 = scan_ctl(argv[1], argc, argv, "INIT_T0", -1, "0", NULL);
t1 = scan_ctl(argv[1], argc, argv, "INIT_T1", -1, "0", NULL);
dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
z0 = scan_ctl(argv[1], argc, argv, "INIT_20", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "INIT_21", -1, "0", NULL);
dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);
land = scan_ctl(argv[1], argc, argv, "INIT_DON", -1, "0", NULL);
00050
00051
00052
00053
00055
              z1 = scan_ctl(argv[1], argc, argv, "INIT_Z1", -1, "0", NULL);
dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);
lon0 = scan_ctl(argv[1], argc, argv, "INIT_LON0", -1, "0", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "INIT_LON1", -1, "0", NULL);
dlon = scan_ctl(argv[1], argc, argv, "INIT_DLON", -1, "1", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "INIT_LAT0", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT1", -1, "0", NULL);
dlat = scan_ctl(argv[1], argc, argv, "INIT_DLAT", -1, "1", NULL);
st = scan_ctl(argv[1], argc, argv, "INIT_ST", -1, "0", NULL);
sz = scan_ctl(argv[1], argc, argv, "INIT_ST", -1, "0", NULL);
slon = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
slat = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
sx = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_UT", -1, "0", NULL);
ulon = scan_ctl(argv[1], argc, argv, "INIT_UT", -1, "0", NULL);
ulon = scan_ctl(argv[1], argc, argv, "INIT_ULON", -1, "0", NULL);
ulon = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
even = (int) scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
m = scan_ctl(argv[1], argc, argv, "INIT_EVENLY", -1, "1", NULL);
even = (int) scan_ctl(argv[1], argc, argv, "INIT_EVENLY", -1, "1", NULL);
m = scan_ctl(argv[1], argc, argv, "INIT_REP", -1, "1", NULL);
00056
00057
00058
00059
00061
00062
00063
00064
00065
00066
00067
00068
00069
00070
00071
00072
00073
00074
00075
00076
               /* Initialize random number generator... */
00077
               gsl rng env setup();
00078
               rng = gsl_rng_alloc(gsl_rng_default);
00079
08000
                 /* Create grid... */
00081
                for (t = t0; t \le t1; t += dt)
00082
                   for (z = z0; z \le z1; z += dz)
                         for (lon = lon0; lon <= lon1; lon += dlon)</pre>
00083
00084
                            for (lat = lat0; lat <= lat1; lat += dlat)</pre>
                                 for (irep = 0; irep < rep; irep++) {</pre>
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)
                                               + gsl_ran_gaussian_ziggurat(rng, DX2DEG(sx, lat) / 2.3548)
+ ulon * (gsl_rng_uniform(rng) - 0.5));
00096
00097
00098
00099
                                         atm->lat[atm->np]
00100
                                             = (lat + gsl_ran_gaussian_ziggurat(rng, slat / 2.3548)
                                     + gsl_ran_gaussian_ziggurat(rng, DY2DEG(sx) / 2.3548)
+ ulat * (gsl_rng_uniform(rng) - 0.5));
} while (even && gsl_rng_uniform(rng) >
00101
00102
00103
00104
                                                        fabs(cos(atm->lat[atm->np] * M_PI / 180.)));
```

```
00106
                   /* Set particle counter... */
00107
                   if ((++atm->np) >= NP)
                     ERRMSG("Too many particles!");
00108
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... */
        if (ctl.qnt_m >= 0)
  for (ip = 0; ip < atm->np; ip++)
00116
00117
00118
            atm->q[ctl.qnt_m][ip] = m / atm->np;
00119
00120
        /* Save data... */
        write_atm(argv[2], &ctl, atm, t0);
00121
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 the Free Software Foundation, either version 3 of the License, or \,
00006
00007
          (at your option) any later version.
00008
00009
          MPTRAC is distributed in the hope that it will be useful,
          but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
00012
          GNU General Public License for more details.
00013
          You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
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
```

5.6 atm init.c 37

```
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;
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
                  /* Read control parameters... */
read_ctl(argv[1], argc, argv, &ctl);
t0 = scan_ctl(argv[1], argc, argv, "INIT_TO", -1, "0", NULL);
t1 = scan_ctl(argv[1], argc, argv, "INIT_T1", -1, "0", NULL);
dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
z0 = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "INIT_20", -1, "0", NULL);
dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);
lon0 = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);
dlon1 = scan_ctl(argv[1], argc, argv, "INIT_LONO", -1, "0", NULL);
dlon2 = scan_ctl(argv[1], argc, argv, "INIT_DLONI", -1, "0", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "INIT_LATO", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT1", -1, "0", NULL);
dlat = scan_ctl(argv[1], argc, argv, "INIT_DLAT", -1, "1", NULL);
st = scan_ctl(argv[1], argc, argv, "INIT_ST", -1, "0", NULL);
sz = scan_ctl(argv[1], argc, argv, "INIT_SZ", -1, "0", NULL);
slon = scan_ctl(argv[1], argc, argv, "INIT_SZ", -1, "0", NULL);
00049
                    /* Read control parameters... */
00050
00051
00052
00053
00054
00055
00056
00057
00059
00060
00061
00062
00063
00064
                   sz = scan_ctl(argv[1], argc, argv, "INIT_SZ", -1, "0", NULL);
slon = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
slat = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
sx = scan_ctl(argv[1], argc, argv, "INIT_SX", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_UT", -1, "0", NULL);
uz = scan_ctl(argv[1], argc, argv, "INIT_UZ", -1, "0", NULL);
ulon = scan_ctl(argv[1], argc, argv, "INIT_ULON", -1, "0", NULL);
ulat = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
00065
00066
00067
00068
00069
00071
                    even = (int) scan_ctl(argv[1], argc, argv, "INIT_EVENLY", -1, "1", NU
rep = (int) scan_ctl(argv[1], argc, argv, "INIT_REP", -1, "1", NULL);
m = scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
00072
00073
00074
00075
00076
                    /* Initialize random number generator... */
00077
                    gsl_rng_env_setup();
00078
                    rng = gsl_rng_alloc(gsl_rng_default);
00079
08000
                     /* Create grid... */
00081
                    for (t = t0; t \le t1; t += dt)
                        for (z = z0; z \le z1; z += dz)
00082
                               for (lon = lon0; lon <= lon1; lon += dlon)</pre>
                                    for (lat = lat0; lat <= lat1; lat += dlat)</pre>
00084
00085
                                          for (irep = 0; irep < rep; irep++) {</pre>
00086
                                               /* Set position... */
00087
00088
                                               atm->time[atm->np]
                                                   = (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]
                                                    + for the first transfer 
00095
00096
00097
00098
                                               do (
00099
                                                    atm->lat[atm->np]
00100
                                                          = (lat + gsl_ran_gaussian_ziggurat(rng, slat / 2.3548)
                                                                   + gsl_ran_gaussian_ziggurat(rng, DY2DEG(sx) / 2.3548)
00101
00102
                                                                   + ulat * (gsl_rng_uniform(rng) - 0.5));
00103
                                               } while (even && gsl_rng_uniform(rng) >
00104
                                                                       fabs(cos(atm->lat[atm->np] \star M_PI / 180.)));
00105
00106
                                                /* Set particle counter... */
                                                       ((++atm->np) >= NP)
00107
                                                      ERRMSG("Too many particles!");
00108
00109
00110
00111
                     /\star Check number of air parcels... \star/
                    if (atm->np <= 0)
00112
                         ERRMSG("Did not create any air parcels!");
00113
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
```

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

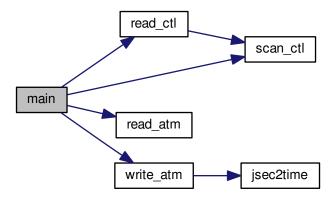
5.7.2.1 int main (int *argc*, char * *argv*[])

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
                 double m, mtot = 0, dt, dx, dz, mmax = 0,
00038
                     t0, t1, z0, z1, lon0, lon1, lat0, lat1;
00039
00040
                 int i, ip, iq, n;
00041
00042
                  /* Allocate... */
00043
                 ALLOC(atm, atm_t, 1);
00044
                 ALLOC(atm2, atm_t, 1);
00045
00046
                  /* Check arguments... ∗/
                  if (argc < 4)
00047
00048
                      ERRMSG("Give parameters: <ctl> <atm_in> <atm_out>");
00049
00050
                 /* Read control parameters... */
                 /* Read control parameters... */
read_ctl(argv[1], argc, argv, &ctl);
n = (int) scan_ctl(argv[1], argc, argv, "SPLIT_N", -1, "", NULL);
m = scan_ctl(argv[1], argc, argv, "SPLIT_M", -1, "-999", NULL);
dt = scan_ctl(argv[1], argc, argv, "SPLIT_DT", -1, "0", NULL);
t0 = scan_ctl(argv[1], argc, argv, "SPLIT_TO", -1, "0", NULL);
t1 = scan_ctl(argv[1], argc, argv, "SPLIT_TI", -1, "0", NULL);
dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);
z0 = scan_ctl(argv[1], argc, argv, "SPLIT_ZO", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "SPLIT_Z1", -1, "0", NULL);
00052
00053
00054
00055
00057
00058
                 z0 = scan_ctl(argv[1], argc, argv, "SPLIT_Z0", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "SPLIT_Z1", -1, "0", NULL);
dx = scan_ctl(argv[1], argc, argv, "SPLIT_DX", -1, "0", NULL);
lon0 = scan_ctl(argv[1], argc, argv, "SPLIT_LON0", -1, "0", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "SPLIT_LON1", -1, "0", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT0", -1, "0", NULL);
00059
00060
00061
00062
```

```
lat1 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT1", -1, "0", NULL);
00064
00065
00066
         /* Init random number generator... */
00067
         gsl_rng_env_setup();
00068
        rng = gsl_rng_alloc(gsl_rng_default);
00069
00070
         /* Read atmospheric data... */
00071
         read_atm(argv[2], &ctl, atm);
00072
00073
         /\star Get total and maximum mass... \star/
00074
         if (ctl.qnt_m >= 0)
          for (ip = 0; ip < atm->np; ip++) {
  mtot += atm->q[ctl.qnt_m][ip];
00075
00076
00077
             mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
00078
00079
        if (m > 0)
08000
          mtot = m;
00081
00082
        /* Loop over air parcels... */
00083
         for (i = 0; i < n; i++) {</pre>
00084
00085
           /* Select air parcel... */
00086
           if (ctl.qnt_m >= 0)
00087
             do {
00088
               ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
             } while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
00089
00090
00091
             ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00092
00093
           /* Set time... */
00094
           if (t1 > t0)
00095
             atm2 \rightarrow time[atm2 \rightarrow np] = t0 + (t1 - t0) * gsl_rng_uniform_pos(rng);
00096
00097
             atm2->time[atm2->np] = atm->time[ip]
00098
                + gsl_ran_gaussian_ziggurat(rng, dt / 2.3548);
00099
00100
           /* Set vertical position... */
00101
           if (z1 > z0)
00102
             atm2 \rightarrow p[atm2 \rightarrow np] = P(z0 + (z1 - z0) * gsl_rng_uniform_pos(rng));
00103
00104
             atm2->p[atm2->np] = atm->p[ip]
               + DZ2DP(gsl_ran_gaussian_ziggurat(rng, dz / 2.3548), atm->p[ip]);
00105
00106
00107
           /* Set horizontal position... */
           if (lon1 > lon0 && lat1 > lat0) {
00108
             atm2->lon[atm2->np] = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
atm2->lat[atm2->np] = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
00109
00110
           } else {
00111
             atm2 \rightarrow lon[atm2 \rightarrow np] = atm \rightarrow lon[ip]
00112
             + gsl_ran_gaussian_ziggurat(rng, DX2DEG(dx, atm->lat[ip]) / 2.3548); atm2->lat[atm2->np] = atm->lat[ip]
00113
00114
00115
               + gsl_ran_gaussian_ziggurat(rng, DY2DEG(dx) / 2.3548);
00116
00117
           /* Copy quantities... */
00118
           for (iq = 0; iq < ctl.nq; iq++)
  atm2->q[iq][atm2->np] = atm->q[iq][ip];
00119
00121
00122
           /* Adjust mass...
           if (ctl.qnt_m >= 0)
00123
00124
             atm2->q[ctl.qnt_m][atm2->np] = mtot / n;
00125
00126
           /* Increment particle counter... */
00127
           if ((++atm2->np) >= NP)
00128
             ERRMSG("Too many air parcels!");
00129
00130
         /* Save data and close file... */
00131
        write_atm(argv[3], &ctl, atm2, atm->time[0]);
00132
00133
00134
         /* Free... */
00135
        free (atm);
00136
        free(atm2);
00137
00138
        return EXIT SUCCESS;
00139 }
```

Here is the call graph for this function:



5.8 atm_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.
80000
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
         You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
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
         double m, mtot = 0, dt, dx, dz, mmax = 0,
00037
00038
           t0, t1, z0, z1, lon0, lon1, lat0, lat1;
00039
00040
         int i, ip, iq, n;
00041
00042
          /* Allocate... */
00043
         ALLOC(atm, atm_t, 1);
00044
         ALLOC(atm2, atm_t, 1);
00045
00046
         /* Check arguments... */
00047
         if (argc < 4)
00048
           ERRMSG("Give parameters: <ctl> <atm_in> <atm_out>");
00049
00050
         /* Read control parameters... */
         read_ctl(argv[1], argc, argv, &ctl);
n = (int) scan_ctl(argv[1], argc, argv, "SPLIT_N", -1, "", NULL);
m = scan_ctl(argv[1], argc, argv, "SPLIT_M", -1, "-999", NULL);
dt = scan_ctl(argv[1], argc, argv, "SPLIT_DT", -1, "0", NULL);
00051
00052
00053
00054
```

5.8 atm split.c 41

```
t0 = scan_ctl(argv[1], argc, argv, "SPLIT_TO", -1, "0", NULL);
          t1 = scan_ctl(argv[1], argc, argv, "SPLIT_TI", -1, "0", NULL);
dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);
00056
00057
         dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);
z0 = scan_ctl(argv[1], argc, argv, "SPLIT_Z0", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "SPLIT_Z1", -1, "0", NULL);
dx = scan_ctl(argv[1], argc, argv, "SPLIT_DX", -1, "0", NULL);
lon0 = scan_ctl(argv[1], argc, argv, "SPLIT_LON0", -1, "0", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "SPLIT_LON1", -1, "0", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT0", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT1", -1, "0", NULL);
00058
00059
00060
00062
00063
00064
00065
00066
          /* Init random number generator... */
00067
         gsl rng env setup();
00068
         rng = gsl_rng_alloc(gsl_rng_default);
00069
00070
          /* Read atmospheric data... */
00071
          read_atm(argv[2], &ctl, atm);
00072
00073
          /\star Get total and maximum mass... \star/
00074
          if (ctl.qnt_m >= 0)
00075
            for (ip = 0; ip < atm->np; ip++) {
00076
               mtot += atm->q[ctl.qnt_m][ip];
00077
               mmax = GSL\_MAX(mmax, atm->q[ctl.qnt_m][ip]);
00078
00079
          if'(m > 0)
00080
            mtot = m;
00081
00082
          /* Loop over air parcels... */
00083
          for (i = 0; i < n; i++) {</pre>
00084
00085
             /* Select air parcel... */
00086
             if (ctl.qnt_m >= 0)
00087
00088
                 ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00089
               } while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
00090
             else
00091
               ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00093
             /* Set time... */
00094
             if (t1 > t0)
00095
              atm2->time[atm2->np] = t0 + (t1 - t0) * gsl_rng_uniform_pos(rng);
00096
             else
              atm2->time[atm2->np] = atm->time[ip]
00097
00098
                  + gsl_ran_gaussian_ziggurat(rng, dt / 2.3548);
00099
             /\star Set vertical position... \star/
00100
00101
             if (z1 > z0)
              atm2->p[atm2->np] = P(z0 + (z1 - z0) * gsl_rng_uniform_pos(rng));
00102
             else
00103
00104
              atm2->p[atm2->np] = atm->p[ip]
00105
                 + DZ2DP(gsl_ran_gaussian_ziggurat(rng, dz / 2.3548), atm->p[ip]);
00106
00107
             /* Set horizontal position...
            if (lon1 > lon0 && lat1 > lat0) {
   atm2->lon[atm2->np] = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
   atm2->lat[atm2->np] = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
00108
00109
00110
00111
             } else {
00112
              atm2->lon[atm2->np] = atm->lon[ip]
00113
                 + gsl_ran_gaussian_ziggurat(rng, DX2DEG(dx, atm->lat[ip]) / 2.3548);
00114
               atm2->lat[atm2->np] = atm->lat[ip]
                  + gsl_ran_gaussian_ziggurat(rng, DY2DEG(dx) / 2.3548);
00115
00116
00117
00118
             /* Copy quantities... */
00119
             for (iq = 0; iq < ctl.nq; iq++)</pre>
00120
              atm2->q[iq][atm2->np] = atm->q[iq][ip];
00121
00122
             /* Adjust mass... */
            if (ctl.qnt_m >= 0)
00123
00124
               atm2->q[ctl.qnt_m][atm2->np] = mtot / n;
00125
00126
             /* Increment particle counter... */
00127
             if ((++atm2->np) >= NP)
00128
               ERRMSG("Too many air parcels!");
00129
00130
          /* Save data and close file... */
00131
00132
          write_atm(argv[3], &ctl, atm2, atm->time[0]);
00133
00134
          /* Free... */
00135
          free (atm);
          free (atm2);
00137
00138
          return EXIT_SUCCESS;
00139 }
```

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

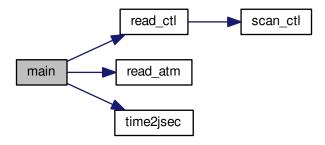
5.9.2.1 int main (int argc, char * argv[])

Definition at line 27 of file atm_stat.c.

```
00029
00030
00031
        ctl_t ctl;
00032
00033
        atm_t *atm;
00034
00035
        FILE *out;
00036
00037
        char tstr[LEN];
00038
00039
        double latm, lats, lonm, lons, t, zm, zs;
00040
00041
        int f, ip, year, mon, day, hour, min;
00042
00043
         /* Allocate... */
00044
        ALLOC(atm, atm_t, 1);
00045
00046
         /* Check arguments... */
00047
        if (argc < 4)
          ERRMSG("Give parameters: <ctl> <outfile> <atml> [<atm2> ...]");
00048
00049
00050
        /* Read control parameters... */
00051
        read_ctl(argv[1], argc, argv, &ctl);
00052
00053
         /* Write info... */
00054
        printf("Write center of mass data: sn'', argv[2]);
00055
00056
        /* Create output file...
        if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");
00057
00058
00059
00060
        /* Write header... */
00061
        00062
                         = time [s]\n"
                  "# $2 = altitude (mean) [km]\n"
00063
00064
                  "# $3 = altitude (sigma) [km] \n"
00065
                  "# $4 = altitude (minimum) [km]n"
                  "# $5 = altitude (10%% percentile) [km]\n"
00066
                  "# $6 = altitude (1st quarter) [km]\n"
"# $7 = altitude (median) [km]\n"
00067
00068
                 "# $8 = altitude (3rd quarter) [km]\n"
"# $9 = altitude (90%% percentile) [km]\n"
00069
00070
00071
                 "# $10 = altitude (maximum) [km]\n");
        fprintf(out,
    "# $11 = longitude (mean) [deg]\n"
    "# $12 = longitude (sigma) [deg]\n"
00072
00073
00074
                  "# $13 = longitude (minimum) [deg]\n"
                 "# $14 = longitude (10%% percentile) [deg]\n"
```

```
"# $15 = longitude (1st quarter) [deg]\n"
00078
                  "# $16 = longitude (median) [deg] n"
                  "# $17 = longitude (3rd quarter) [deg]\n"
00079
                  "# $18 = longitude (90%% percentile) [deg]\n"
00080
                  "# $19 = longitude (maximum) [deg] \n");
00081
00082
        fprintf(out,
                  "# $20 = latitude (mean) [deg]\n
00084
                  "# $21 = latitude (sigma) [deg] n"
00085
                  "# $22 = latitude (minimum) [deg] n"
                  "# $23 = latitude (10%% percentile) [deg]\n"
00086
                  "# $24 = latitude (1st quarter) [deg] n"
00087
                  "# $25 = latitude (median) [deg] n"
00088
00089
                  "# $26 = latitude (3rd quarter) [deg]\n"
                  "# $27 = latitude (90%% percentile) [deg]\n"
00090
00091
                  "# $28 = latitude (maximum) [deg] \n\n");
00092
00093
        /* Loop over files... */
00094
        for (f = 3; f < argc; f++) {</pre>
00095
00096
           /* Read atmopheric data... */
00097
           read_atm(argv[f], &ctl, atm);
00098
00099
           /* Initialize... */
00100
           zm = zs = 0;
           lonm = lons = 0;
00101
           latm = lats = 0;
00102
00103
00104
           /* Calculate mean and standard deviation... */
00105
           for (ip = 0; ip < atm->np; ip++) {
00106
             zm += Z(atm->p[ip]) / atm->np;
              lonm += atm->lon[ip] / atm->np;
00107
00108
             latm += atm->lat[ip] / atm->np;
00109
              zs += SQR(Z(atm->p[ip])) / atm->np;
             lons += SQR(atm->lon[ip]) / atm->np;
lats += SQR(atm->lat[ip]) / atm->np;
00110
00111
00112
00113
00114
           /* Normalize... */
00115
           zs = sqrt(zs - SQR(zm));
           lons = sqrt(lons - SQR(lonm));
lats = sqrt(lats - SQR(latm));
00116
00117
00118
           /* Sort arrays... */
gsl_sort(atm->p, 1, (size_t) atm->np);
gsl_sort(atm->lon, 1, (size_t) atm->np);
gsl_sort(atm->lat, 1, (size_t) atm->np);
00119
00120
00121
00122
00123
           00124
00125
00126
           year = atoi(tstr);
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00127
00128
           mon = atoi(tstr);
00129
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
           day = atoi(tstr);
sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00130
00131
           hour = atoi(tstr);
00132
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00134
           min = atoi(tstr);
00135
           time2jsec(year, mon, day, hour, min, 0, 0, &t);
00136
           00137
00138
00139
00140
                    Z(atm->p[atm->np - atm->np / 10]),
Z(atm->p[atm->np - atm->np / 4]),
00141
00142
                    Z(atm->p[atm->np - atm->np / 4],
Z(atm->p[atm->np / 2]), Z(atm->p[atm->np / 4]),
Z(atm->p[atm->np / 10]), Z(atm->p[0]),
lonm, lons, atm->lon[0], atm->lon[atm->np / 10],
00143
00144
00145
00146
                    atm->lon[atm->np / 4], atm->lon[atm->np / 2],
                    atm->lon[atm->np - atm->np / 4],
atm->lon[atm->np - atm->np / 10],
00147
00148
                    atm->lon[atm->np - 1],
00149
                    latm, lats, atm->lat[0], atm->lat[atm->np / 10],
00150
                    atm->lat(atm->np / 4], atm->lat(atm->np / 2], atm->lat(atm->np - atm->np / 4],
00151
00153
                    atm->lat[atm->np - atm->np / 10], atm->lat[atm->np - 1]);
00154
00155
        /* Close file... */
00156
00157
        fclose(out);
00158
00159
         /* Free... */
00160
        free(atm);
00161
00162
         return EXIT_SUCCESS;
00163 }
```

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 along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00015
00016
00017
        Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
        int argc,
00028
00029
        char *argv[]) {
00030
00031
        ctl_t ctl;
00032
00033
        atm_t *atm;
00034
00035
        FILE *out;
00036
00037
        char tstr[LEN];
00038
00039
        double latm, lats, lonm, lons, t, zm, zs;
00040
00041
        int f, ip, year, mon, day, hour, min;
00042
00043
         /* Allocate... */
00044
        ALLOC(atm, atm_t, 1);
00045
00046
         /* Check arguments... */
00047
         if (argc < 4)
00048
          ERRMSG("Give parameters: <ctl> <outfile> <atml> [<atm2> ...]");
00049
00050
         /\star Read control parameters... \star/
00051
        read_ctl(argv[1], argc, argv, &ctl);
00052
00053
         /* Write info... */
00054
        printf("Write center of mass data: %s\n", argv[2]);
00055
00056
        /* Create output file... */
if (!(out = fopen(argv[2], "w")))
00057
00058
          ERRMSG("Cannot create file!");
00059
```

5.10 atm stat.c 45

```
/* Write header... */
00061
        fprintf(out,
00062
                  "# $1 = time [s] \n"
                 "# $2 = altitude (mean) [km]\n"
00063
                  "# $3 = altitude (sigma) [km] \n"
00064
00065
                 "# $4 = altitude (minimum) [km]\n"
                        = altitude (10%% percentile) [km]\n"
                 "# $5
                 "# $6
00067
                        = altitude (1st quarter) [km]\n"
00068
                 "# $7 = altitude (median) [km]\n"
                 "# $8 = altitude (3rd quarter) [km]\n"
"# $9 = altitude (90%% percentile) [km]\n"
00069
00070
                 "# $10 = altitude (maximum) [km]\n");
00071
00072
        fprintf(out,
00073
                  "# $11 = longitude (mean) [deg]\n"
00074
                 "# $12 = longitude (sigma) [deg]\n"
                 "# $13 = longitude (minimum) [deg]\n"
00075
                  "# $14 = longitude (10%% percentile) [deg]\n"
00076
00077
                 "# $15 = longitude (1st quarter) [deg] \n"
                 "# $16 = longitude (median) [deg]\n"
00079
                 "# $17 = longitude (3rd quarter) [deg]\n"
00080
                 "# $18 = longitude (90%% percentile)
                 "# $19 = longitude (maximum) [deg]\n");
00081
        fprintf(out, "# $20 = latitude (mean) [deg]\n
00082
00083
00084
                 "# $21 = latitude (sigma) [deg]\n"
                 "# $22 = latitude (minimum) [deg]\n"
00086
                 "# $23 = latitude (10%% percentile) [deg]\n"
00087
                 "# $24 = latitude (1st quarter) [deg]\n"
                  "# $25 = latitude (median) [deg] n"
00088
                 "# $26 = latitude (3rd quarter) [deg]\n"
"# $27 = latitude (90%% percentile) [deg]\n"
00089
00090
00091
                 "# $28 = latitude (maximum) [deg] \n\n");
00092
        /* Loop over files... */
00093
00094
        for (f = 3; f < argc; f++) {</pre>
00095
00096
           /* Read atmopheric data... */
          read_atm(argv[f], &ctl, atm);
00098
00099
           /* Initialize... */
00100
          zm = zs = 0;
          lonm = lons = 0;
latm = lats = 0;
00101
00102
00103
00104
           /* Calculate mean and standard deviation... */
00105
           for (ip = 0; ip < atm->np; ip++) {
00106
            zm += Z(atm->p[ip]) / atm->np;
             lonm += atm->lon[ip] / atm->np;
latm += atm->lat[ip] / atm->np;
00107
00108
             zs += SQR(Z(atm->p[ip])) / atm->np;
lons += SQR(atm->lon[ip]) / atm->np;
00109
00110
             lats += SQR(atm->lat[ip]) / atm->np;
00111
00112
00113
          /* Normalize... */
zs = sqrt(zs - SQR(zm));
00114
00115
           lons = sqrt(lons - SQR(lonm));
00117
           lats = sqrt(lats - SQR(latm));
00118
00119
           /* Sort arrays... */
          gsl_sort(atm->p, 1, (size_t) atm->np);
gsl_sort(atm->lon, 1, (size_t) atm->np);
gsl_sort(atm->lat, 1, (size_t) atm->np);
00120
00121
00122
00123
00124
           /\star Get time from filename... \star/
00125
           sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00126
           year = atoi(tstr);
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00127
00128
          mon = atoi(tstr);
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00130
           day = atoi(tstr);
00131
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00132
          hour = atoi(tstr);
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00133
00134
           min = atoi(tstr);
00135
           time2jsec(year, mon, day, hour, min, 0, 0, &t);
00136
          00137
00138
                    00139
                    t, zm, zs, Z(atm->p[atm->np - 1]),
00140
                    Z(atm->p[atm->np - atm->np / 10]),
Z(atm->p[atm->np - atm->np / 4]),
00141
00142
                    Z(atm \rightarrow p[atm \rightarrow np / 2]), Z(atm \rightarrow p[atm \rightarrow np / 4]),
00143
                   Z(atm->p[atm->np / 10]), Z(atm->p[0]), lonm, lons, atm->lon[0], atm->lon[atm->np / 10],
00144
00145
00146
                    atm->lon[atm->np / 4], atm->lon[atm->np / 2],
```

```
atm->lon[atm->np - atm->np / 4],
atm->lon[atm->np - atm->np / 10],
atm->lon[atm->np - 1],
00149
                         latm, lats, atm->lat[0], atm->lat[atm->np / 10],
00150
                        atm->lat[atm->np / 4], atm->lat[atm->np / 2],
atm->lat[atm->np - atm->np / 4],
atm->lat[atm->np - atm->np / 10], atm->lat[atm->np - 1]);
00151
00152
00153
00154
00155
00156
           /* Close file... */
00157
          fclose(out);
00158
           /* Free... */
00159
00160
          free(atm);
00161
00162
           return EXIT_SUCCESS;
00163 }
```

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

Here is the call graph for this function:



5.12 day2doy.c 47

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.
80000
00009
        MPTRAC is distributed in the hope that it will be useful,
        but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
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 <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>.
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)</pre>
00035
           ERRMSG("Give parameters: <year> <mon> <day>");
00036
00037
        /* Read arguments... */
        year = atoi(argv[1]);
mon = atoi(argv[2]);
00038
00039
00040
        day = atoi(argv[3]);
00041
        /* Convert... */
day2doy(year, mon, day, &doy);
00042
00043
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
         /* Check arguments... */
00033
00034
         if (argc < 3)
00035
          ERRMSG("Give parameters: <year> <doy>");
00036
00037
        /* Read arguments... */
00038
        vear = atoi(argv[1]);
00039
        doy = atoi(argv[2]);
00040
00041
         /* Convert... */
        doy2day(year, doy, &mon, &day);
printf("%d %d %d\n", year, mon, day);
00042
00043
00044
00045
        return EXIT_SUCCESS;
00046 }
```

Here is the call graph for this function:



5.14 doy2day.c

```
00001 /*
         This file is part of MPTRAC.
00002
00003
         MPTRAC is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by
00004
00005
00006
          the Free Software Foundation, either version 3 of the License, or
00007
          (at your option) any later version.
80000
         MPTRAC is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00009
00010
00011
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 <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/>.
00016
00017
          Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028
         int argc,
00029
         char *argv[]) {
00030
00031
          int day, doy, mon, year;
00032
00033
          /* Check arguments... */
00034
          if (argc < 3)
00035
            ERRMSG("Give parameters: <year> <doy>");
00036
```

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

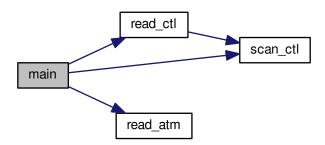
5.15.2.1 int main (int argc, char * argv[])

Definition at line 27 of file extract.c.

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

```
fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00063
        ctl.qnt_unit[iq]);
fprintf(out, "\n");
00064
00065
00066
        /* Loop over files... */
for (f = 3; f < argc; f++) {</pre>
00067
00068
00070
           /* Read atmopheric data... */
          if (!(in = fopen(argv[f], "r")))
00071
00072
            continue;
00073
          else
00074
           fclose(in);
00075
          read_atm(argv[f], &ctl, atm);
00076
          00077
00078
00079
                  Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
          for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");</pre>
08000
00081
00082
             fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00083
00084
          fprintf(out, "\n");
00085
00086
00087
        /* Close file... */
00088
        fclose(out);
00089
        /* Free... */
00090
00091
        free(atm);
00092
00093
        return EXIT SUCCESS:
00094 }
```

Here is the call graph for this function:



5.16 extract.c

```
00001 /*
00002
         This file is part of MPTRAC.
00003
00004
          \ensuremath{\mathsf{MPTRAC}} is free software: you can redistribute it and/or modify
          it under the terms of the GNU General Public License as published by
00005
          the Free Software Foundation, either version 3 of the License, or
00006
00007
          (at your option) any later version.
80000
         MPTRAC is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00009
00010
00011
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 <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>.
00016
00017
          Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
```

```
00025 #include "libtrac.h"
00027 int main(
00028 int argc,
00029
       char *argv[]) {
00030
       ctl_t ctl;
00032
00033
       atm_t *atm;
00034
00035
       FILE *in, *out;
00036
00037
        int f, ip, iq;
00038
        /* Allocate... */
00039
00040
       ALLOC(atm, atm_t, 1);
00041
00042
        /* Check arguments... */
00043
        if (argc < 4)
00044
          ERRMSG("Give parameters: <ctl> <outfile> <atml> [<atm2> ...]");
00045
00046
       /* Read control parameters... */
        read_ctl(argv[1], argc, argv, &ctl);
00047
        ip = (int) scan_ctl(argv[1], argc, argv, "EXTRACT_IP", -1, "0", NULL);
00048
00049
00050
        /* Write info... */
00051
        printf("Write trajectory data: sn", argv[2]);
00052
       /* Create output file... */
if (!(out = fopen(argv[2], "w")))
00053
00054
00055
         ERRMSG("Cannot create file!");
00056
00057
        /* Write header... */
00058
       fprintf(out,
       00059
00060
00061
00062
00063
       ctl.qnt_unit[iq]);
fprintf(out, "\n");
00064
00065
00066
        /* Loop over files... */
for (f = 3; f < argc; f++) {</pre>
00067
00068
00069
00070
          /★ Read atmopheric data... ★/
00071
          if (!(in = fopen(argv[f], "r")))
00072
            continue;
00073
          else
00074
           fclose(in);
00075
          read_atm(argv[f], &ctl, atm);
00076
         /* Write data... */
fprintf(out, "%.2f %g %g %g", atm->time[ip],
00077
00078
                  Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
00079
          for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");</pre>
08000
00082
            fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00083
00084
          fprintf(out, "\n");
00085
00086
00087
        /* Close file... */
00088
       fclose(out);
00089
        /* Free... */
00090
00091
       free(atm);
00092
00093
        return EXIT_SUCCESS;
00094 }
```

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 %g\n", year, mon, day, hour, min, sec, remain);
00045
00046
        return EXIT SUCCESS:
00047 }
```

Here is the call graph for this function:



5.18 jsec2time.c

```
00001 /*
00002
         This file is part of MPTRAC.
00003
00004
         MPTRAC is free software: you can redistribute it and/or modify
         it under the terms of the GNU General Public License as published by
00005
         the Free Software Foundation, either version 3 of the License, or
00006
00007
         (at your option) any later version.
80000
00009
         \ensuremath{\mathsf{MPTRAC}} is distributed in the hope that it will be useful,
         but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
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 <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>.
00016
00017
         Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
```

```
00025 #include "libtrac.h"
00027 int main(
       int argc,
00028
00029
       char *argv[]) {
00030
       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 intpol_met_2d (double array[EX][EY], int ix, int iy, double wx, double wy, double *var)

Linear interpolation of 2-D meteorological data.

- void intpol_met_3d (float array[EX][EY][EP], int ip, int ix, int iy, double wp, double wx, double wy, double *var)

 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 *pt, 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. void 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. • void 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 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) 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 *lon)
```

Convert Cartesian coordinates to geolocation.

Definition at line 29 of file libtrac.c.

```
00033 {
00034
00035 double radius;
00036
00037 radius = NORM(x);
00038 *lat = asin(x[2] / radius) * 180 / M_PI;
00039 *lon = atan2(x[1], x[0]) * 180 / M_PI;
00040 *z = radius - RE;
00041 }
```

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

Climatology of HNO3 volume mixing ratios.

Definition at line 45 of file libtrac.c.

```
00048
00049
00050
          static double secs[12] = { 1209600.00, 3888000.00, 6393600.00,
00051
             9072000.00, 11664000.00, 14342400.00,
             16934400.00, 19612800.00, 22291200.00, 24883200.00, 27561600.00, 30153600.00
00052
00053
00054
00055
00056
          static double lats[18] = { -85, -75, -65, -55, -45, -35, -25, -15, -5,
00057
             5, 15, 25, 35, 45, 55, 65, 75, 85
00058
00059
00060
          static double ps[10] = { 4.64159, 6.81292, 10, 14.678, 21.5443,
00061
             31.6228, 46.4159, 68.1292, 100, 146.78
00062
00063
00064
          static double hno3[12][18][10] = {
00065
             {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00066
               {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
               \{0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54\},
00067
00068
                {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00069
               \{0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709\},
               {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37}, {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244}, {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
00070
00071
00072
00073
               {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181}
               {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104}, {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
00074
00075
               {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185}, {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745}, {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00076
00077
00078
               {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
08000
                {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00081
                {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14}
00082
                {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
              {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00083
               {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
00084
               {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
{0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00085
00086
               {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661}, {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
00087
00088
               {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184}, {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189}, {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
00089
00090
00091
00092
               {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
00093
                {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
               {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191}, {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875}, {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11}, {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
00094
00095
00096
00097
00098
               {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
```

```
{1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
             {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}, {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00100
00101
00102
               \{0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52\},
               {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3}, {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
00103
00104
               {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
               {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33}
00106
00107
               {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174}
00108
               {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
               \{0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186\},
00109
               {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
00110
00111
               \{0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135\},
               {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
00112
00113
               \{0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1\},\
               {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12}, {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82}, {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54}, {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08}, {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
00114
00115
00116
00118
              {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
00119
00120
               {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00121
               \{1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5\},\
               {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09}, {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
00122
00123
               {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
               {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
00125
00126
               {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12}
00127
               \{0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172\},
00128
               {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
00129
               {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
00130
               \{0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286\},\
               {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02}
00131
00132
               {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
00133
               {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
00134
               \{0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04\},
               {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
00135
               {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
00137
              {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
               {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57}, {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63}, {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37}, {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88}, {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
00138
00139
00140
00141
00142
               {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
00144
               {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
00145
               \{0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126\},
               {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183}, {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18}, {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343}, {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
00146
00147
00148
               {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83}, {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25}, {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
00150
00151
00152
             {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52}, {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}}, {1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
00153
00154
               \{1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05\},\
00156
               {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
00157
00158
               {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
00159
               {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
00160
               {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
               {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
00162
               {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
00163
               {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
               {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194}, {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229}, {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
00164
00165
00166
               {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
00167
               {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41}, {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
00169
00170
               {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9}
               {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88}, {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}},
00171
00172
             {3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33}, {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
00173
               {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
00175
               {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3}
{1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
00176
00177
               {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
00178
               {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
00179
               {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
               {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12}, {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
00181
00182
00183
               {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25}
               {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259}, {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
00184
00185
```

```
{0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
             {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4}, {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
00188
00189
              \{0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61\},
            {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}}, {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
00190
00191
             {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
              {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
00193
00194
              {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14}
             {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38}, {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
00195
00196
              {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
00197
00198
              \{0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181\},
              {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
00199
00200
              {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
00201
              {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
00202
              {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
              {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
00203
              \{0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754\},
00205
              {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
              {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
00206
00207
              {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5}
00208
              \{0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55\}\}
            {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56}, {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
00209
00210
              \{0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09\},\
              {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
00212
00213
              {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22}
             {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646}, {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
00214
00215
              {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
00216
              \{0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815\},
              {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
00218
00219
              {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
             {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163}, {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
00220
00221
             {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714}, {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1}, {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
00222
00224
              {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
00225
00226
              {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
00227
            \{\{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91\},
              {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
00228
              {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
             {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56}, {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
00231
00232
              {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
00233
              {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
              {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
00234
              {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
00235
              \{0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146\},
              {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
00237
00238
              {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
             {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353}, {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
00239
00240
             {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2}, {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
00241
              {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
00243
00244
              {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
00245
            {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78}
             {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73}, {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
00246
00247
00248
              {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
              \{0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955\},
00249
00250
              {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61}
00251
              {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
00252
              {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
              \{0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121\},
00253
              {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
              \{0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146\},
              {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172}, {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
00256
00257
             {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
{1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
00258
00259
              {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
00260
              {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
              {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
00262
            {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89}
{0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
00263
00264
00265
              \{0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65\},
              {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
00266
              {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45,
                                                                                  0.837}
              {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488}, {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
00268
00269
             {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198}, {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173}, {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
00270
00271
00272
```

```
{0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
                {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189}, {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
00274
00275
                {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
00276
                {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85}, {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
00277
00278
                {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35}, {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
00280
00281
00282
00283
           double aux00, aux01, aux10, aux11, sec;
00284
00285
           int ilat, ip, isec;
00286
00287
           /\star Get seconds since begin of year... \star/
00288
           sec = fmod(t, 365.25 * 86400.);
00289
00290
           /* Get indices... */
           isec = locate_irr(secs, 12, sec);
00291
00292
           ilat = locate_reg(lats, 18, lat);
00293
           ip = locate_irr(ps, 10, p);
00294
00295
           /* Interpolate... */
           00296
00297
           aux01 = LIN(ps[ip], hno3[isec][ilat + 1][ip],
00298
00299
                            ps[ip + 1], hno3[isec][ilat + 1][ip + 1], p);
          ps[ip + 1], hno3[isec][ilat + 1][ip + 1], p);

aux10 = LIN(ps[ip], hno3[isec + 1][ilat][ip],

ps[ip + 1], hno3[isec + 1][ilat][ip + 1], p);

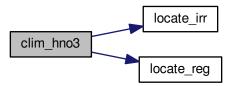
aux11 = LIN(ps[ip], hno3[isec + 1][ilat + 1][ip],

ps[ip + 1], hno3[isec + 1][ilat + 1][ip + 1], p);

aux00 = LIN(lats[ilat], aux00, lats[ilat + 1], aux01, lat);

aux11 = LIN(lats[ilat], aux10, lats[ilat + 1], aux11, lat);
00300
00301
00302
00303
00304
00305
00306
           return LIN(secs[isec], aux00, secs[isec + 1], aux11, sec);
00307 }
```

Here is the call graph for this function:



5.19.2.3 double clim_tropo (double t, double lat)

Climatology of tropopause pressure.

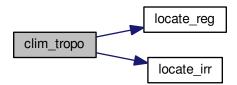
Definition at line 311 of file libtrac.c.

```
00314
00315
              static double doys[12]
              = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00316
00317
              static double lats[73]
00318
                  = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5, -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5, -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
00320
00321
                  -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5, 15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5, 45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5, 75, 77.5, 80, 82.5, 85, 87.5, 90
00322
00323
00324
00325
00326
```

```
00328
             static double tps[12][73]
00329
                = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
00330
                         297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
00331
                         175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
                         99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54, 98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5,
00332
                         152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
00334
00335
                         277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
             275.3, 275.6, 275.4, 274.1, 273.5}, {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7, 300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
00336
00337
00338
               150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
00339
               98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
00340
00341
               98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
00342
               220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
               284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
00343
00344
               287.5, 286.2, 285.8},
             {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
               297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
00346
               161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
00347
00348
               100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
00349
               99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
               186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8, 279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
00350
00351
               304.3, 304.9, 306, 306.6, 306.2, 306},
              {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
00353
00354
               290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
               195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7, 102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79, 99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
00355
00356
00357
               148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6,
               263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.
00359
00360
               315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
             {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8, 260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9, 205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7, 101.8, 101.4, 101.1, 101, 101, 101.1, 101.2, 101.5, 101.9,
00361
00362
00363
00365
               102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
               165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
00366
00367
               273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
               325.3, 325.8, 325.8},
00368
             $22.3, $23.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $25.8, $2
00369
00370
00371
00372
00373
               106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
00374
               127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
             251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9, 308.5, 312.2, 313.1, 313.3}, {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
00375
00376
               187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
00378
00379
               235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
              110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7, 111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
00380
00381
               117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9, 224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
00382
               275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
00384
             {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
00385
00386
               185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
00387
               233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
               110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9, 112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
00388
               120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
00390
00391
               230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7
00392
               278.2, 282.6, 287.4, 290.9, 292.5, 293},
00393
             {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
               183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
00394
               243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
00395
               114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4,
               110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
00397
00398
               114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
              203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5, 276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
00399
00400
             {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5, 215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
00401
               237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7,
00403
                                                                                                  124.8,
00404
               111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
00405
               106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
               112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
00406
               206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
00407
               279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
               305.1},
00409
              {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
00410
00411
               253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
               223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9, 108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
00412
00413
```

```
102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
              109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4, 241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
00415
00416
            286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6}, {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7, 284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192, 175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1, 100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
00417
00418
00419
00420
00421
             100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160, 186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278, 280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4, 281.7, 281.1, 281.2}
00422
00423
00424
00425
00426
00427
00428
            double doy, p0, p1;
00429
00430
            int imon, ilat:
00431
            /* Get day of year... */
doy = fmod(t / 86400., 365.25);
00432
00433
00434
            while (doy < 0)
00435
               doy += 365.25;
00436
00437
            /* Get indices... */
00438
            ilat = locate_reg(lats, 73, lat);
            imon = locate_irr(doys, 12, doy);
00439
00440
00441
             /* Interpolate...
00442
            p0 = LIN(lats[ilat], tps[imon][ilat],
            00443
00444
00445
00446
            return LIN(doys[imon], p0, doys[imon + 1], p1, doy);
00447 }
```

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 451 of file libtrac.c.

```
00455
00456
00457
        int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
        int d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00458
00459
00460
        if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
  *doy = d01[mon - 1] + day - 1;
00461
00462
00463
        else
00464
           *doy = d0[mon - 1] + day - 1;
00465 }
```

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

Get date from day of year.

Definition at line 469 of file libtrac.c.

```
00473
00474
          int d0[12] = \{ 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 \};
int d01[12] = \{ 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 \};
00475
00476
00478
00479
           /\star Get month and day... \star/
          if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
  for (i = 11; i >= 0; i--)
   if (d01[i] <= doy)</pre>
00480
00481
00482
             break;
*mon = i + 1;
00483
00484
00485
             *day = doy - d01[i] + 1;
00486
          } else {
           for (i = 11; i >= 0; i--)
if (d0[i] <= doy)
00487
00488
00489
                  break;
00490
             *mon = i + 1;
00491
             *day = doy - d0[i] + 1;
00492 }
00493 }
```

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

Convert geolocation to Cartesian coordinates.

Definition at line 497 of file libtrac.c.

```
00501 {
00502
00503 double radius;
00504
00505 radius = z + RE;
00506 x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
00507 x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
00508 x[2] = radius * sin(lat / 180 * M_PI);
00509 }
```

5.19.2.7 void get_met (ctl t * ctl, char * metbase, double t, met t ** met0, met t ** met1)

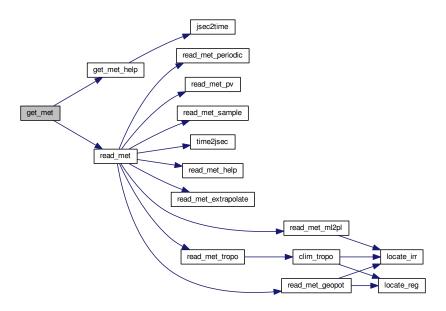
Get meteorological data for given timestep.

Definition at line 513 of file libtrac.c.

```
{
00519
00520
       static int init, ip, ix, iy;
00521
00522
       met t *mets:
00523
00524
       char filename[LEN];
00525
00526
        /* Init... */
       if (t == ctl->t_start || !init) {
00527
00528
         init = 1;
00529
00530
         get_met_help(t, -1, metbase, ctl->dt_met, filename);
00531
         read_met(ctl, filename, *met0);
00532
00533
         get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
     dt_met, filename);
    read_met(ctl, filename, *met1);
00534
00535 }
00536
```

```
/* Read new data for forward trajectories... */
        if (t > (*met1)->time && ct1->direction == 1) {
  mets = *met1;
  *met1 = *met0;
00538
00539
00540
           *met0 = mets;
00541
00542
           get_met_help(t, 1, metbase, ctl->dt_met, filename);
00543
           read_met(ctl, filename, *metl);
00544
00545
00546
         /* Read new data for backward trajectories... */
00547
         if (t < (*met0)->time && ctl->direction == -1) {
00548
          mets = *met1;
*met1 = *met0;
00549
00550
           *met0 = mets;
00551
           get_met_help(t, -1, metbase, ctl->dt_met, filename);
00552
           read_met(ctl, filename, *met0);
00553
00554
00555
        /* Check that grids are consistent... */
00556
         if ((*met0)->nx != (*met1)->nx
00557
              || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
00558
          ERRMSG("Meteo grid dimensions do not match!");
        for (ix = 0; ix < (*met0)->nx; ix++)
  if ((*met0)->lon[ix] != (*met1)->lon[ix])
00559
00560
         ERRMSG("Meteo grid longitudes do not match!");
for (iy = 0; iy < (*met0)->ny; iy++)
00561
00562
00563
              ((*met0)->lat[iy] != (*met1)->lat[iy])
00564
             ERRMSG("Meteo grid latitudes do not match!");
         for (ip = 0; ip < (*met0)->np; ip++)
  if ((*met0)->p[ip] != (*met1)->p[ip])
00565
00566
00567
             ERRMSG("Meteo grid pressure levels do not match!");
00568 }
```

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)

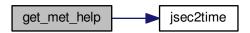
Get meteorological data for timestep.

Definition at line 572 of file libtrac.c.

```
00577
00578
00579 double t6, r;
```

```
00581
        int year, mon, day, hour, min, sec;
00582
00583
        /\star Round time to fixed intervals... \star/
00584
        if (direct == -1)
          t6 = floor(t / dt_met) * dt_met;
00585
00586
00587
          t6 = ceil(t / dt_met) * dt_met;
00588
00589
        /* Decode time... */
00590
        jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00591
        /* Set filename... */
sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", metbase, year, mon, day, hour);
00592
00593
00594 }
```

Here is the call graph for this function:



5.19.2.9 void intpol_met_2d (double array[EX][EY], int ix, int iy, double wx, double wy, double * var)

Linear interpolation of 2-D meteorological data.

Definition at line 598 of file libtrac.c.

```
00604
00605
00606
         double aux00, aux01, aux10, aux11;
00607
00608
         /* Set variables... */
00609
        aux00 = array[ix][iy];
         aux01 = array[ix][iy + 1];
aux10 = array[ix + 1][iy];
00610
00611
00612
         aux11 = array[ix + 1][iy + 1];
00613
00614
         /* Interpolate horizontally... */
        aux00 = wy * (aux00 - aux01) + aux01;
aux11 = wy * (aux10 - aux11) + aux11;
00615
00616
         *var = wx * (aux00 - aux11) + aux11;
00618 }
```

5.19.2.10 void intpol_met_3d (float array[EX][EY][EP], int ip, int ix, int iy, double wp, double wx, double wy, double * var)

Linear interpolation of 3-D meteorological data.

Definition at line 622 of file libtrac.c.

```
00630
00631
00632
        double aux00, aux01, aux10, aux11;
00633
00634
         /* Interpolate vertically... */
00635
        aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00636
          + array[ix][iy][ip + 1];
00637
        aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
        + array[ix][iy + 1][ip + 1];
aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
00638
00639
00640
          + array[ix + 1][iy][ip + 1];
00641
        aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
00642
           + array[ix + 1][iy + 1][ip + 1];
00643
00644
        /* Interpolate horizontally... */
        aux00 = wy * (aux00 - aux01) + aux01;
aux11 = wy * (aux10 - aux11) + aux11;
00645
00646
00647
        *var = wx * (aux00 - aux11) + aux11;
00648 }
```

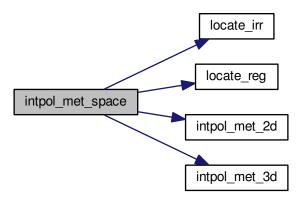
5.19.2.11 void intpol_met_space (met_t * met, double p, double lon, double lon, double * ps, double *

Spatial interpolation of meteorological data.

Definition at line 652 of file libtrac.c.

```
00666
                      {
00667
00668
        double wp, wx, wy;
00670
        int ip, ix, iy;
00671
        /* Check longitude... */
if (met->lon[met->nx - 1] > 180 && lon < 0)
00672
00673
00674
         lon += 360;
00675
00676
        /* Get indices... */
00677
        ip = locate_irr(met->p, met->np, p);
        ix = locate_reg(met->lon, met->nx, lon);
00678
00679
        iy = locate_reg(met->lat, met->ny, lat);
00680
00682
        wp = (met - p[ip + 1] - p) / (met - p[ip + 1] - met - p[ip]);
        wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);
00683
00684
00685
00686
        /* Interpolate... */
00687
        if (ps != NULL)
00688
          intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
00689
        if (pt != NULL)
00690
         intpol_met_2d(met->pt, ix, iy, wx, wy, pt);
        if (z != NULL)
00691
00692
          intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy, z);
00693
        if (t != NULL)
00694
          intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
00695
        if (u != NULL)
00696
          intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
        if (v != NULL)
00697
00698
          intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
00699
        if (w != NULL)
00700
          intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
00701
        if (pv != NULL)
00702
          intpol_met_3d(met->pv, ip, ix, iy, wp, wx, wy, pv);
        if (h2o != NULL)
00703
00704
          intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
        if (o3 != NULL)
00706
          intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00707 }
```

Here is the call graph for this function:



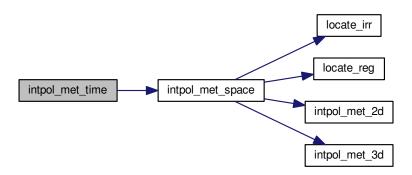
5.19.2.12 void intpol_met_time (met_t * met0, met_t * met1, double ts, double p, double lon, double lat, double * ps, double * pt, double * z, double * t, double * u, double * v, double * w, double * pv, double * h2o, double * o3)

Temporal interpolation of meteorological data.

Definition at line 711 of file libtrac.c.

```
00727
00728
00729
       double h2o0, h2o1, o30, o31, ps0, ps1, pt0, pt1, pv0, pv1, t0, t1, u0, u1,
00730
         v0, v1, w0, w1, wt, z0, z1;
00731
00732
       /* Spatial interpolation... */
       00733
00734
00735
                        pt == NULL ? NULL : &pt0,
00736
                        z == NULL ? NULL : &z0,
00737
                        t == NULL ? NULL : &t0,
00738
                        u == NULL ? NULL : &u0,
                        v == NULL ? NULL : &v0,
00739
00740
                        w == NULL ? NULL : &w0,
00741
                        pv == NULL ? NULL : &pv0,
00742
                        h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
       00743
00744
00745
00746
                        z == NULL ? NULL : &z1,
00747
                        t == NULL ? NULL : &t1,
00748
                        u == NULL ? NULL : &u1,
00749
                        v == NULL ? NULL : &v1,
00750
                        w == NULL ? NULL : &w1,
                        pv == NULL ? NULL : &pv1,
00751
00752
                        h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00753
00754
       /* Get weighting factor... */
       wt = (met1->time - ts) / (met1->time - met0->time);
00755
00756
00757
       /* Interpolate... */
00758
       if (ps != NULL)
         *ps = wt * (ps0 - ps1) + ps1;
00760
       if (pt != NULL)
       *pt = wt * (pt0 - pt1) + pt1;
if (z != NULL)
00761
00762
00763
         *z = wt * (z0 - z1) + z1;
       if (t != NULL)
00764
         *t = wt * (t0 - t1) + t1;
00765
00766
       if (u != NULL)
00767
       *u = wt * (u0 - u1) + u1;
if (v != NULL)
00768
00769
         *v = wt * (v0 - v1) + v1;
00770
       if (w != NULL)
00771
         *w = wt * (w0 - w1) + w1;
00772
       if (pv != NULL)
00773
         *pv = wt * (pv0 - pv1) + pv1;
       if (h2o != NULL)
00774
00775
         *h2o = wt * (h2o0 - h2o1) + h2o1;
       if (03 != NULL)
00776
00777
         *o3 = wt * (o30 - o31) + o31;
00778 }
```

Here is the call graph for this function:



5.19.2.13 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 782 of file libtrac.c.

```
00790
00791
00792
         struct tm t0, *t1;
00793
         time_t jsec0;
00795
00796
         t0.tm_year = 100;
         t0.tm\_mon = 0;
00797
00798
         t0.tm_mday = 1;
t0.tm_hour = 0;
t0.tm_min = 0;
00799
00800
00801
         t0.tm\_sec = 0;
00802
00803
         jsec0 = (time_t) jsec + timegm(&t0);
00804
        t1 = gmtime(&jsec0);
00805
00806
         *year = t1->tm_year + 1900;
         *mon = t1->tm_mon + 1;
*day = t1->tm_mday;
00807
80800
00809
         *hour = t1->tm_hour;
         *min = t1->tm_min;
*sec = t1->tm_sec;
*remain = jsec - floor(jsec);
00810
00811
00812
00813 }
```

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

Find array index for irregular grid.

Definition at line 817 of file libtrac.c.

```
00820 {
00821
00822 int i, ilo, ihi;
00823
00824 ilo = 0;
00825 ihi = n - 1;
00826 i = (ihi + ilo) >> 1;
```

```
if (xx[i] < xx[i + 1])
        while (ihi > ilo + 1) {
  i = (ihi + ilo) >> 1;
00830
           if (xx[i] > x)
00831
             ihi = i;
00832
            else
00833
             ilo = i;
00835
       } else
00836
        while (ihi > ilo + 1) {
            i = (ihi + ilo) >> 1;
00837
           if (xx[i] <= x)
00838
             ihi = i;
00839
            else
00840
00841
             ilo = i;
        }
00842
00843
00844
        return ilo;
00845 }
```

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

Find array index for regular grid.

Definition at line 849 of file libtrac.c.

```
00852
                   {
00853
00855
        /* Calculate index... */
i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
00856
00857
00858
00859
        /* Check range... */
        if (i < 0)
00860
00861
          i = 0;
00862
        else if (i \ge n - 2)
00863
         i = n - 2;
00864
00865
        return i:
00866 }
```

5.19.2.16 void read_atm (const char * filename, ctl_t * ctl, atm_t * atm)

Read atmospheric data.

Definition at line 870 of file libtrac.c.

```
00873
00874
00875
        FILE *in;
00876
00877
        char line[LEN], *tok;
00878
00879
        double t0;
00880
00881
        int dimid, ip, iq, ncid, varid;
00882
00883
        size_t nparts;
00884
        /* Init... */
00885
        atm->np = 0;
00886
00887
88800
        /* Write info... */
         printf("Read atmospheric data: sn", filename);
00889
00890
        /* Read ASCII data... */
if (ctl->atm_type == 0) {
00891
00892
00893
          /* Open file... */
if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");
00894
00895
00896
00897
00898
           /* Read line... */
00899
           while (fgets(line, LEN, in)) {
```

```
00900
              /* Read data... */
TOK(line, tok, "%lg", atm->time[atm->np]);
TOK(NULL, tok, "%lg", atm->p[atm->np]);
TOK(NULL, tok, "%lg", atm->lon[atm->np]);
TOK(NULL, tok, "%lg", atm->lat[atm->np]);
for (iq = 0; iq < ctl->nq; iq++)
TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00901
00902
00903
00904
00905
00907
00908
               /* Convert altitude to pressure... */
atm->p[atm->np] = P(atm->p[atm->np]);
00909
00910
00911
               /* Increment data point counter... */
if ((++atm->np) > NP)
00912
00913
00914
                 ERRMSG("Too many data points!");
00915
00916
00917
            /* Close file... */
00918
            fclose(in);
00919
00920
00921
          /* Read binary data... */
         else if (ctl->atm_type == 1) {
00922
00923
00924
            /* Open file... */
            if (!(in = fopen(filename, "r")))
00925
               ERRMSG("Cannot open file!");
00926
00927
00928
             /* Read data... */
00929
            FREAD(&atm->np, int,
00930
                    1.
00931
                    in);
00932
            FREAD(atm->time, double,
00933
                      (size_t) atm->np,
00934
                    in);
            FREAD(atm->p, double,
00935
00936
                      (size_t) atm->np,
                    in);
00938
            FREAD(atm->lon, double,
00939
                      (size_t) atm->np,
00940
                    in);
00941
            FREAD (atm->lat, double,
00942
                      (size_t) atm->np,
                    in);
00943
00944
            for (iq = 0; iq < ctl->nq; iq++)
00945
              FREAD(atm->q[iq], double,
00946
                         (size_t) atm->np,
00947
                      in);
00948
00949
            /* Close file... */
00950
            fclose(in);
00951
00952
00953
          /* Read netCDF data... */
00954
         else if (ctl->atm_type == 2) {
00955
00956
             /* Open file... */
00957
            NC(nc_open(filename, NC_NOWRITE, &ncid));
00958
            /* Get dimensions... */
NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
NC(nc_inq_dimlen(ncid, dimid, &nparts));
00959
00960
00961
            atm->np = (int) nparts;
if (atm->np > NP)
00962
00963
00964
               ERRMSG("Too many particles!");
00965
00966
            /* Get time... */
            NC(nc_inq_varid(ncid, "time", &varid));
00967
00968
            NC(nc_get_var_double(ncid, varid, &t0));
            for (ip = 0; ip < atm->np; ip++)
  atm->time[ip] = t0;
00970
00971
            /* Read geolocations... */
NC(nc_inq_varid(ncid, "PRESS", &varid));
00972
00973
            NC(nc_get_var_double(ncid, varid, atm->p));
NC(nc_inq_varid(ncid, "LON", &varid));
00974
00975
            NC(nc_get_var_double(ncid, varid, atm->lon));
NC(nc_inq_varid(ncid, "LAT", &varid));
00976
00977
00978
            NC(nc_get_var_double(ncid, varid, atm->lat));
00979
00980
            /* Read variables... */
            if (ctl->qnt_p >= 0)
00982
               if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
00983
                 NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
            if (ctl->qnt_t >= 0)
00984
               if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
00985
00986
                 NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
```

```
if (ctl->qnt_u >= 0)
00988
             if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
00989
                NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
00990
           if (ctl->qnt_v >= 0)
             if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
00991
           NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
if (ctl->qnt_w >= 0)
00992
00994
                 (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
00995
               NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
00996
           if (ct1->qnt h2o >= 0)
           if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
    NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
if (ctl->qnt_o3 >= 0)
00997
00998
00999
             if (nc_inq_varid(ncid, "03", &varid) == NC_NOERR)
01000
01001
                NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
           if (ctl->qnt_theta >= 0)
  if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
01002
01003
           NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
if (ctl->qnt_pv >= 0)
01004
01005
                 (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
01007
                NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
01008
           /* Check data... */
for (ip = 0; ip < atm->np; ip++)
   if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
01009
01010
01011
                  || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
01012
01013
                   \label{eq:ctl-qnt_h2o} $$ | (ctl->qnt_h2o) = 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)$
                  || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > le10)
|| (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > le10)) {
01014
01015
01016
                atm->time[ip] = GSL_NAN;
                atm->p[ip] = GSL_NAN;
01017
01018
                atm->lon[ip] = GSL_NAN;
01019
                atm->lat[ip] = GSL_NAN;
01020
                for (iq = 0; iq < ctl->nq; iq++)
01021
                  atm->q[iq][ip] = GSL_NAN;
01022
             } else {
01023
                if (ct1->qnt h2o >= 0)
                  atm->q[ctl->qnt_h2o][ip] *= 1.608;
01025
                if (ctl->qnt_pv >= 0)
01026
                  atm->q[ctl->qnt_pv][ip] *= 1e6;
01027
                if (atm->lon[ip] > 180)
                  atm->lon[ip] -= 360;
01028
            }
01029
01030
01031
            /* Close file... */
01032
           NC(nc_close(ncid));
01033
01034
         /* Error... */
01035
01036
           ERRMSG("Atmospheric data type not supported!");
01038
01039
         /\star Check number of points... \star/
01040
         if (atm->np < 1)
           ERRMSG("Can not read any data!");
01041
01042 }
```

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

Read control parameters.

Definition at line 1046 of file libtrac.c.

```
01050
                 {
01051
01052
      int ip, iq;
01053
      01055
01056
            argv[0], __DATE__, __TIME__);
01057
01058
01059
      /* Initialize quantity indices... */
01060
      ctl->qnt_ens = -1;
01061
      ctl->qnt_m = -1;
      ctl->qnt_r = -1;
01062
01063
      ctl->qnt_rho = -1;
01064
      ctl->qnt_ps = -1;
01065
      ctl->qnt_pt = -1;
01066
      ctl->qnt_z = -1;
```

```
01067
       ctl->qnt_p = -1;
        ctl->qnt_t = -1;
01068
        ctl->qnt_u = -1;
01069
        ctl->qnt_v = -1;
01070
        ctl->qnt_w = -1;
01071
01072
        ctl \rightarrow qnt_h2o = -1;
        ct1->qnt_o3 = -1;
01073
01074
        ctl->qnt\_theta = -1;
01075
        ctl->qnt_vh = -1;
01076
        ctl->qnt_vz = -1;
        ctl->qnt_pv = -1;
01077
01078
        ctl->qnt\_tice = -1;
01079
        ctl->qnt_tsts = -1;
01080
        ctl->qnt_tnat = -1;
01081
        ctl->qnt_stat = -1;
01082
01083
        /* Read quantities... */
        ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
01084
        if (ctl->nq > NQ)
01085
01086
          ERRMSG("Too many quantities!");
        for (iq = 0; iq < ctl->nq; iq++) {
01087
01088
          01089
01090
01091
01092
01093
          /* Try to identify quantity... */
if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
01094
01095
01096
            ctl->qnt_ens = iq;
01097
            sprintf(ctl->qnt_unit[iq], "-");
01098
          } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
01099
             ctl->qnt_m = iq;
01100
             sprintf(ctl->qnt_unit[iq], "kg");
01101
           } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
01102
            ctl->qnt_r = iq;
             sprintf(ctl->qnt_unit[iq], "m");
01103
          } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
01105
             ctl->qnt_rho = iq;
01106
             sprintf(ctl->qnt_unit[iq], "kg/m^3");
          } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
  ctl->qnt_ps = iq;
  sprintf(ctl->qnt_unit[iq], "hPa");
01107
01108
01109
          } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
01110
             ctl->qnt_pt = iq;
01111
01112
            sprintf(ctl->qnt_unit[iq], "hPa");
01113
          } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
            ctl->qnt_z = iq;
01114
            sprintf(ctl->qnt_unit[iq], "km");
01115
01116
          } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
            ctl->qnt_p = iq;
01117
01118
            sprintf(ctl->qnt_unit[iq], "hPa");
01119
          } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
            ctl->qnt_t = iq;
sprintf(ctl->qnt_unit[iq], "K");
01120
01121
          } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
  ctl->qnt_u = iq;
01122
01124
            sprintf(ctl->qnt_unit[iq], "m/s");
01125
          } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
            ctl->qnt_v = iq;
01126
             sprintf(ctl->qnt_unit[iq], "m/s");
01127
          } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
01128
            ctl->qnt_w = iq;
            sprintf(ctl->qnt_unit[iq], "hPa/s");
01130
01131
           } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
01132
             ctl->qnt_h2o = iq;
             sprintf(ctl->qnt_unit[iq], "1");
01133
          } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
  ctl->qnt_o3 = iq;
01134
01135
            sprintf(ctl->qnt_unit[iq], "1");
01137
            else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
01138
             ctl->qnt_theta = iq;
            sprintf(ctl->qnt_unit[iq], "K");
01139
          } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
  ctl->qnt_vh = iq;
  sprintf(ctl->qnt_unit[iq], "m/s");
01140
01141
01142
01143
          } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
             ctl->qnt_vz = iq;
01144
            sprintf(ctl->qnt_unit[iq], "m/s");
01145
          } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
  ctl->qnt_pv = iq;
  sprintf(ctl->qnt_unit[iq], "PVU");
01146
01147
01148
          } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
01149
01150
            ctl->qnt_tice = iq;
            sprintf(ctl->qnt_unit[iq], "K");
01151
          } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
ctl->qnt_tsts = iq;
01152
01153
```

```
sprintf(ctl->qnt_unit[iq], "K");
           } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
01155
             ctl->qnt_tnat = iq;
01156
             sprintf(ctl->qnt_unit[iq], "K");
01157
           } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
ctl->qnt_stat = iq;
01158
01159
01160
             sprintf(ctl->qnt_unit[iq], "-");
01161
01162
             scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
01163
01164
01165
         /* Time steps of simulation... */
01166
        ctl->direction
           (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
01167
01168
            (ctl->direction != -1 && ctl->direction != 1)
01169
          ERRMSG("Set DIRECTION to -1 or 1!");
        ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
01170
01171
01172
01173
         /* Meteorological data... */
01174
         ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
         ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL); ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
01175
01176
         ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL); ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
01177
01178
         ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
01179
         ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
01180
         ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
01181
01182
         if (ctl->met_np > EP)
          ERRMSG("Too many levels!");
01183
01184
         for (ip = 0; ip < ctl->met_np; ip++)
01185
           ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
         ctl->met_tropo
01186
01187
           = (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "0", NULL);
        scan_ctl(filename, argc, argv, "MET_GEOPOT", -1, "-", ctl->met_geopot);
scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
01188
01189
01190
        ctl->met dt out =
01191
           scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
01192
01193
         /* Isosurface parameters... */
01194
        ctl->isosurf
        = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
01195
01196
01197
01198
         /* Diffusion parameters... */
01199
         ctl->turb_dx_trop
01200
          = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
01201
         ctl->turb dx strat
           = scan ctl(filename, argc, argv, "TURB DX STRAT", -1, "0", NULL);
01202
01203
         ctl->turb_dz_trop
01204
            = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
01205
         ctl->turb_dz_strat
01206
           = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
01207
        ctl->turb mesox =
           scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
01208
01209
        ctl->turb mesoz =
01210
           scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
01211
01212
         /* Mass and life time..
        ctl->molmass = scan_ctl(filename, argc, argv, "MOLMASS", -1, "1", NULL);
01213
01214
        ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
         ctl->tdec_strat =
01215
01216
           scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
01217
01218
         /* PSC analysis... */
01219
         ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
        ctl->psc hno3 =
01220
01221
           scan ctl(filename, argc, argv, "PSC HNO3", -1, "9e-9", NULL);
01222
        /* Output of atmospheric data... */
scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
01224
      atm_basename);
01225
         scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
01226
         ctl->atm dt out
           scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
01227
01228
         ctl->atm_filter
           (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
01229
01230
        ctl->atm_type =
           (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
01231
01232
        /* Output of CSI data... */
01233
01234
        scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi basename);
01235
        ctl->csi_dt_out =
        scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->
01236
01237
      csi_obsfile);
```

```
01238
        ctl->csi obsmin =
           scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
01239
01240
         ctl->csi_modmin =
01241
           scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
         ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
01242
01243
01244
01245
         ctl->csi_lon0 =
         scan_ctl(filename, argc, argv, "CSI_LONO", -1, "-180", NULL);
ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
01246
01247
01248
         ctl->csi nx =
         (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL)
ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
01249
01250
01251
01252
         ctl->csi_ny =
01253
           (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
01254
01255
         /* Output of ensemble data... */
         scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
01256
      ens_basename);
01257
01258
         /* Output of grid data... */
         scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
01259
                    ctl->grid basename);
01260
         scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
01261
      grid_gpfile);
         ctl->grid_dt_out =
01262
01263
           scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
01264
         ctl->grid_sparse
01265
           (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
         ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
01266
01267
01268
01269
           (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
01270
         ctl->grid_lon0 =
           scan_ctl(filename, argc, argv, "GRID_LONO", -1, "-180", NULL);
01271
01272
         ctl->grid lon1 =
01273
           scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
01274
         ctl->grid_nx =
01275
            (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
01276
         ctl->grid_lat0 =
           scan_ctl(filename, argc, argv, "GRID_LATO", -1, "-90", NULL);
01277
01278
         ct1->grid lat1
01279
           scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
         ctl->grid_ny =
01280
01281
            (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
01282
01283
         /* Output of profile data... */
         01284
01285
01286
         scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
      prof_obsfile);
         ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
01287
01288
01289
         ctl->prof_nz =
01290
            (int) scan ctl(filename, argc, argv, "PROF NZ", -1, "60", NULL);
01291
         ctl->prof_lon0 =
01292
           scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
01293
         ctl->prof_lon1 =
01294
           scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
01295
         ctl->prof nx =
01296
           (int) scan ctl(filename, argc, argv, "PROF NX", -1, "360", NULL);
01297
         ctl->prof_lat0 =
01298
            scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
         ctl->prof_lat1 =
01299
01300
           scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
01301
         ctl->prof_ny =
           (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
01302
01303
01304
         /* Output of station data... */
01305
         scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
01306
                    ctl->stat basename);
         ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
01307
01308
01309
```

Here is the call graph for this function:



```
5.19.2.18 void read_met ( ctl_t * ctl, char * filename, met_t * met )
```

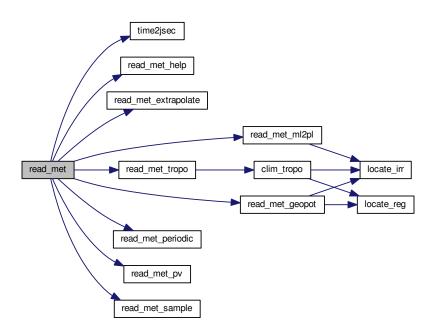
Read meteorological data file.

Definition at line 1314 of file libtrac.c.

```
01317
01318
01319
        char cmd[2 * LEN], levname[LEN], tstr[10];
01320
01321
        static float help[EX * EY];
01322
        int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
01324
01325
        size_t np, nx, ny;
01326
        /* Write info... */
01327
01328
        printf("Read meteorological data: %s\n", filename);
01329
01330
         /\star Get time from filename... \star/
01331
        sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
        year = atoi(tstr);
01332
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
01333
01334
        mon = atoi(tstr);
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
01335
01336
        day = atoi(tstr);
01337
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
01338
        hour = atoi(tstr);
01339
        time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
01340
        /* Open netCDF file... */
01341
01342
        if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
01343
01344
          /* Try to stage meteo file...
01345
          if (ctl->met_stage[0] != '-') {
   sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
01346
             year, mon, day, hour, filename);
if (system(cmd) != 0)
01347
01348
01349
               ERRMSG("Error while staging meteo data!");
01350
01351
           /* Try to open again... */
01352
          NC(nc_open(filename, NC_NOWRITE, &ncid));
01353
01354
01355
         /* Get dimensions... */
01356
        NC(nc_inq_dimid(ncid, "lon", &dimid));
NC(nc_inq_dimlen(ncid, dimid, &nx));
if (nx < 2 || nx > EX)
01357
01358
01359
01360
          ERRMSG("Number of longitudes out of range!");
01361
01362
        NC(nc_inq_dimid(ncid, "lat", &dimid));
01363
        NC(nc_inq_dimlen(ncid, dimid, &ny));
        if (ny < 2 || ny > EY)
    ERRMSG("Number of latitudes out of range!");
01364
01365
01366
        sprintf(levname, "lev");
01367
01368
        NC(nc_inq_dimid(ncid, levname, &dimid));
01369
        NC(nc_inq_dimlen(ncid, dimid, &np));
01370
        if (np == 1) {
01371
          sprintf(levname, "lev_2");
01372
          NC(nc_inq_dimid(ncid, levname, &dimid));
01373
          NC(nc_inq_dimlen(ncid, dimid, &np));
```

```
01374
         if (np < 2 || np > EP)
01375
01376
           ERRMSG("Number of levels out of range!");
01377
01378
         /* Store dimensions... */
         met->np = (int) np;
met->nx = (int) nx;
01379
01380
01381
         met->ny = (int) ny;
01382
         /* Get horizontal grid... */
NC(nc_inq_varid(ncid, "lon", &varid));
01383
01384
         NC(nc_get_var_double(ncid, varid, met->lon));
NC(nc_inq_varid(ncid, "lat", &varid));
01385
01386
01387
         NC(nc_get_var_double(ncid, varid, met->lat));
01388
         /* Read meteorological data... */
read_met_help(ncid, "t", "T", met, met->t, 1.0);
read_met_help(ncid, "u", "U", met, met->u, 1.0);
read_met_help(ncid, "v", "V", met, met->v, 1.0);
01389
01390
01391
01392
         read_met_help(ncid, "w", "W", met, met->w, 0.01f);
read_met_help(ncid, "q", "Q", met, met->help(ncid, "q", "Q", met, met->help(ncid, "o3", "03", met, met->o3, (float) (MA / 48.00));
01393
01394
01395
01396
01397
         /* Meteo data on pressure levels... */
01398
         if (ctl->met_np <= 0) {</pre>
01399
01400
            /* Read pressure levels from file...
01401
           NC(nc_inq_varid(ncid, levname, &varid));
01402
           NC(nc\_get\_var\_double(ncid, varid, met->p));
01403
            for (ip = 0; ip < met->np; ip++)
              met->p[ip] /= 100.;
01404
01405
01406
            /* Extrapolate data for lower boundary... */
01407
            read_met_extrapolate(met);
01408
01409
         /* Meteo data on model levels... */
01410
01411
         else {
01412
           /* Read pressure data from file... */
read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
01413
01414
01415
            /\star Interpolate from model levels to pressure levels... \star/
01416
01417
            read_met_ml2pl(ctl, met, met->t);
            read_met_ml2pl(ctl, met, met->u);
01418
01419
            read_met_ml2pl(ctl, met, met->v);
01420
            read_met_ml2pl(ctl, met, met->w);
01421
            read_met_ml2pl(ctl, met, met->h2o);
01422
            read_met_ml2pl(ctl, met, met->o3);
01423
01424
            /* Set pressure levels... */
           met->np = ctl->met_np;
for (ip = 0; ip < met->np; ip++)
01425
01426
01427
              met->p[ip] = ctl->met_p[ip];
01428
01429
01430
         /* Check ordering of pressure levels... */
         for (ip = 1; ip < met->np; ip++)
  if (met->p[ip - 1] < met->p[ip])
01431
01432
01433
              ERRMSG("Pressure levels must be descending!");
01434
01435
         01436
01437
01438
           NC(nc_get_var_float(ncid, varid, help));
01439
            for (iy = 0; iy < met->ny; iy++)
              for (ix = 0; ix < met \rightarrow nx; ix++)
01440
         met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
} else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR
01441
01442
                      || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
01443
01444
            NC(nc_get_var_float(ncid, varid, help));
01445
            for (iy = 0; iy < met->ny; iy++)
              for (ix = 0; ix < met->nx; ix++)
01446
                met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
01447
01448
         } else
01449
           for (ix = 0; ix < met->nx; ix++)
01450
              for (iy = 0; iy < met->ny; iy++)
01451
                met->ps[ix][iy] = met->p[0];
01452
01453
         /* Create periodic boundary conditions... */
01454
         read met periodic (met);
01455
01456
         /* Calculate geopotential heights... */
01457
         read_met_geopot(ctl, met);
01458
01459
         /* Calculate potential vorticity... */
01460
         read_met_pv(met);
```

Here is the call graph for this function:



5.19.2.19 void read_met_extrapolate (met_t * met)

Extrapolate meteorological data at lower boundary.

Definition at line 1474 of file libtrac.c.

```
01475
01477
           int ip, ip0, ix, iy;
01478
01479
           /* Loop over columns... */
01480 #pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
01481 for (ix = 0; ix < met->nx; ix++)
01482
             for (iy = 0; iy < met->ny; iy++) {
01483
01484
                 /\star Find lowest valid data point... \star/
                 for (ip0 = met->np - 1; ip0 >= 0; ip0--)
   if (!gsl_finite(met->t[ix][iy][ip0])
01485
01486
                          | !gsl_finite(met->u[ix][iy][ip0])
| !gsl_finite(met->v[ix][iy][ip0])
01487
01488
01489
                          || !gsl_finite(met->w[ix][iy][ip0]))
01490
01491
                 /* Extrapolate... */
for (ip = ip0; ip >= 0; ip--) {
  met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
  met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
01492
01493
01494
01495
```

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

Calculate geopotential heights.

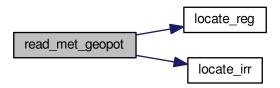
Definition at line 1506 of file libtrac.c.

```
01508
01509
01510
       static double topo_lat[EY], topo_lon[EX], topo_z[EX][EY];
01512
       static int init, topo_nx = -1, topo_ny;
01513
01514
       FILE *in:
01515
01516
       char line[LEN];
01517
01518
       double data[30], lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;
01519
01520
       float help[EX][EY];
01521
01522
       int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;
01524
       /* Initialize geopotential heights... */
01525 #pragma omp parallel for default(shared) private(ix,iy,ip)
01526
       for (ix = 0; ix < met->nx; ix++)
         for (iy = 0; iy < met->ny; iy++)
  for (ip = 0; ip < met->np; ip++)
01527
01528
01529
             met->z[ix][iy][ip] = GSL_NAN;
01531
       /* Check filename... */
       if (ctl->met_geopot[0] == '-')
01532
01533
         return:
01534
01535
       /* Read surface geopotential... */
01536
       if (!init) {
01537
         init = 1;
01538
         /* Write info... */
01539
01540
         printf("Read surface geopotential: %s\n", ctl->met geopot);
01541
01542
          /* Open file... */
         if (!(in = fopen(ctl->met_geopot, "r")))
    ERRMSG("Cannot open file!");
01543
01544
01545
01546
         /* Read data... */
         while (fgets(line, LEN, in))
01547
01548
           if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rz) == 3) {
01549
              if (rlon != rlon_old)
01550
               if ((++topo_nx) >= EX)
01551
                 ERRMSG("Too many longitudes!");
               topo_ny = 0;
01552
01553
             rlon_old = rlon;
01555
              topo_lon[topo_nx] = rlon;
              topo_lat[topo_ny] = rlat;
01556
01557
              topo_z[topo_nx][topo_ny] = rz;
01558
             if ((++topo_ny) >= EY)
               ERRMSG("Too many latitudes!");
01559
01560
01561
          if ((++topo_nx) >= EX)
01562
           ERRMSG("Too many longitudes!");
01563
          /* Close file... */
01564
01565
         fclose(in);
01566
01567
          /* Check grid spacing... */
         01568
01569
01570
            printf("Warning: Grid spacing does not match!\n");
01571
01572
01573
       /\star Apply hydrostatic equation to calculate geopotential heights... \star/
```

```
01574 #pragma omp parallel for default(shared) private(ix,iy,lon,lat,tx,ty,z0,z1,ip0,ts,ip)
       for (ix = 0; ix < met->nx; ix++)
01575
01576
          for (iy = 0; iy < met->ny; iy++) {
01577
01578
            /* Get surface height... */
01579
            lon = met->lon[ix];
            if (lon < topo_lon[0])</pre>
01580
01581
              lon += 360;
            else if (lon > topo_lon[topo_nx - 1])
lon -= 360;
01582
01583
            lat = met->lat[iy];
01584
01585
            tx = locate_reg(topo_lon, topo_nx, lon);
            ty = locate_reg(topo_lat, topo_ny, lat);
01586
            z0 = LIN(topo_lon[tx], topo_z[tx][ty],
topo_lon[tx + 1], topo_z[tx + 1][ty], lon);
01587
01588
            01589
01590
01591
            z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);
01592
01593
            /* Find surface pressure level... */
01594
            ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
01595
            01596
01597
01598
01599
            /* Upper part of profile... */
01600
01601
            met->z[ix][iy][ip0 + 1]
              = (float) (z0 + RI / MA / G0 * 0.5 * (ts + met->t[ix][iy][ip0 + 1])

* log(met->ps[ix][iy] / met->p[ip0 + 1]));
01602
01603
            for (ip = ip0 + 2; ip < met->np; ip++)
01604
01605
              met->z[ix][iy][ip]
01606
                = (float) (met->z[ix][iy][ip - 1] + RI / MA / G0
01607
                            * 0.5 * (met \rightarrow t[ix][iy][ip - 1] + met \rightarrow t[ix][iy][ip])
                            * log(met->p[ip - 1] / met->p[ip]));
01608
01609
         }
01610
01611
        /* Smooth fields... */
01612 #pragma omp parallel for default(shared) private(ip,ix,iy,n,ix2,ix3,iy2,data)
01613
       for (ip = 0; ip < met->np; ip++) {
01614
          /* Median filter... */
01615
          for (ix = 0; ix < met->nx; ix++)
01616
01617
            for (iy = 0; iy < met->nx; iy++) {
01618
             n = 0;
01619
              for (ix2 = ix - 2; ix2 \le ix + 2; ix2++) {
01620
               ix3 = ix2;
                if (ix3 < 0)
01621
01622
                  ix3 += met->nx;
01623
                if (ix3 >= met -> nx)
                  ix3 -= met->nx;
01624
01625
                for (iy2 = GSL_MAX(iy - 2, 0); iy2 <= GSL_MIN(iy + 2, met->ny - 1);
01626
                      iy2++)
01627
                  if (gsl_finite(met->z[ix3][iy2][ip])) {
01628
                    data[n] = met -> z[ix3][iy2][ip];
01629
                    n++;
01630
01631
01632
              if (n > 0) {
                gsl_sort(data, 1, (size_t) n);
help[ix][iy] = (float)
01633
01634
01635
                  gsl_stats_median_from_sorted_data(data, 1, (size_t) n);
01636
              } else
01637
                help[ix][iy] = GSL_NAN;
01638
01639
01640
          /* Copy data... */
          for (ix = 0; ix < met->nx; ix++)
for (iy = 0; iy < met->nx; iy++)
01641
01642
              met \rightarrow z[ix][iy][ip] = help[ix][iy];
01643
01644
01645 }
```

Here is the call graph for this function:



5.19.2.21 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 1649 of file libtrac.c.

```
01655
01657
        static float help[EX * EY * EP];
01658
01659
        int ip, ix, iy, varid;
01660
        /* Check if variable exists... */
if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
01661
01662
          if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
01664
01665
        /* Read data... */
NC(nc_get_var_float(ncid, varid, help));
01666
01667
01668
01669
         /* Copy and check data... */
01670 #pragma omp parallel for default(shared) private(ix,iy,ip)
01671
        for (ix = 0; ix < met->nx; ix++)
          for (iy = 0; iy < met->ny; iy++)
  for (ip = 0; ip < met->np; ip++) {
01672
01673
               dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
01674
01675
                if (fabsf(dest[ix][iy][ip]) < le14f)</pre>
01676
                  dest[ix][iy][ip] *= scl;
01677
01678
                  dest[ix][iy][ip] = GSL_NAN;
01679
01680 }
```

5.19.2.22 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 1684 of file libtrac.c.

```
for (ip = 0; ip < met->np; ip++)
01700
                p[ip] = met->pl[ix][iy][ip];
01701
               /* Interpolate... */
01702
               for (ip = 0; ip < ctl->met_np; ip++) {
01703
                pt = ctl->met_p[ip];
01704
01705
                 if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
01706
                   pt = p[0];
01707
                 else if ((pt > p[met->np - 1] && p[1] > p[0])
                 || (pt < p[met->np - 1] && p[1] < p[0]))
pt = p[met->np - 1];
ip2 = locate_irr(p, met->np, pt);
01708
01709
01710
                 aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
01711
01712
01713
01714
              /* Copy data... */
for (ip = 0; ip < ctl->met_np; ip++)
  var[ix][iy][ip] = (float) aux[ip];
01715
01716
01717
01718
01719 }
```

Here is the call graph for this function:



5.19.2.23 void read_met_periodic (met_t * met)

Create meteorological data with periodic boundary conditions.

Definition at line 1723 of file libtrac.c.

```
01724
01726
           int ip, iy;
01727
01728
            /* Check longitudes... */
            if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
01729
                             + \text{ met} - \ln[1] - \text{ met} - \ln[0] - 360) < 0.01)
01730
01731
01733
           /* Increase longitude counter... */
01734
           if ((++met->nx) > EX)
01735
              ERRMSG("Cannot create periodic boundary conditions!");
01736
01737
           /* Set longitude... */
          met \rightarrow lon[met \rightarrow nx - 1] = met \rightarrow lon[met \rightarrow nx - 2] + met \rightarrow lon[1] - met \rightarrow
01738
01739
01740
           /* Loop over latitudes and pressure levels... */
01741 #pragma omp parallel for default(shared) private(iy,ip)
01742 for (iy = 0; iy < met->ny; iy++) {
01743 met->ps[met->nx - 1][iy] = met->ps[0][iy];
01744 met->pt[met->nx - 1][iy] = met->pt[0][iy];
01745
              for (ip = 0; ip < met->np; ip++) {
                 met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
01746
01747
01748
01749
                 met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
                 met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];

met->pv[met->nx - 1][iy][ip] = met->pv[0][iy][ip];

met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
01750
01751
01752
                 met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01753
01754
01755
           }
01756 }
```

```
5.19.2.24 void read_met_pv ( met_t * met )
```

Calculate potential vorticity.

Definition at line 1760 of file libtrac.c.

```
01761
                        {
01762
01763
         double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
01764
          dtdp, dudp, dvdp, latr, vort, pows[EP];
01765
01766
        int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
01767
01768
         /* Set powers... */
01769
        for (ip = 0; ip < met->np; ip++)
01770
           pows[ip] = pow(1000. / met->p[ip], 0.286);
01771
01772 /* Loop over grid points... */
01773 #pragma omp parallel for default(shared)
       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)
01774
         for (ix = 0; ix < met->nx; ix++) {
01775
           /* Set indices... */
ix0 = GSL_MAX(ix - 1, 0);
01776
01777
           ix1 = GSL_MIN(ix + 1, met -> nx - 1);
01778
01780
           /* Loop over grid points... */
01781
           for (iy = 0; iy < met->ny; iy++) {
01782
             /* Set indices... */
iy0 = GSL_MAX(iy - 1, 0);
01783
01784
             iy1 = GSL_MIN(iy + 1, met->ny - 1);
01785
01786
01787
              /* Set auxiliary variables... */
01788
             latr = GSL_MIN(GSL_MAX(met->lat[iy], -89.), 89.);
             01789
01790
             c0 = cos(met->lat[iy0] / 180. * M_PI);
c1 = cos(met->lat[iy1] / 180. * M_PI);
01791
01792
01793
              cr = cos(latr / 180. * M_PI);
01794
             vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
01795
01796
             /* Loop over grid points... */
for (ip = 0; ip < met->np; ip++) {
01797
01798
01799
                /* Get gradients in longitude... */
                01800
01801
01802
01803
                /* Get gradients in latitude... */
                dtdy = (met \rightarrow t[ix][iy1][ip] - met \rightarrow t[ix][iy0][ip]) * pows[ip] / dy;
01804
                dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
01805
01806
                /* Set indices... */
01807
01808
                ip0 = GSL\_MAX(ip - 1, 0);
                ip1 = GSL_MIN(ip + 1, met->np - 1);
01809
01810
01811
                /* Get gradients in pressure... */
                dp0 = 100. * (met->p[ip] - met->p[ip0]);
dp1 = 100. * (met->p[ip1] - met->p[ip]);
01812
01813
01814
                if (ip != ip0 && ip != ip1) {
                 denom = dp0 * dp1 * (dp0 + dp1);
dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
01815
01816
                          - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
01818
                           + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
01819
                    / denom;
                  01820
01821
                           + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
01822
01823
                    / denom;
                  dvdp = (dp0 * dp0 * met -> v[ix][iy][ip1]
01824
                          - dp1 * dp1 * met->v[ix][iy][ip0]
+ (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
01825
01826
                    / denom;
01827
                } else {
01828
01829
                  denom = dp0 + dp1;
01830
                  dtdp =
                 (met->t[ix][iy][ip1] * pows[ip1] -
   met->t[ix][iy][ip0] * pows[ip0]) / denom;
dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
01831
01832
01833
01834
01835
01836
```

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

Downsampling of meteorological data.

Definition at line 1848 of file libtrac.c.

```
01850
01851
         met_t *help;
01853
01854
         float w, wsum;
01855
01856
         int ip, ip2, ix, ix2, ix3, iy, iy2;
01857
01858
         /* Check parameters... */
01859
         01860
              && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
01861
01862
          /* Allocate... */
01863
         ALLOC(help, met_t, 1);
01865
01866
         /* Copy data... */
         help->nx = met->nx;
help->ny = met->ny;
01867
01868
01869
         help->np = met->np;
01870
         memcpy(help->lon, met->lon, sizeof(met->lon));
01871
         memcpy(help->lat, met->lat, sizeof(met->lat));
01872
         memcpy(help->p, met->p, sizeof(met->p));
01873
01874
         /* Smoothing... */
         for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
01875
           for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
01877
              for (ip = 0; ip < met->np; ip += ctl->met_dp) {
                help->ps[ix][iy] = 0;
help->pt[ix][iy] = 0;
01878
01879
                 help->z[ix][iy][ip] = 0;
01880
                 help->t[ix][iy][ip] = 0;
01881
                 help \rightarrow u[ix][iy][ip] = 0;
01882
                 help \rightarrow v[ix][iy][ip] = 0;
01884
                 help->w[ix][iy][ip] = 0;
01885
                 help \rightarrow pv[ix][iy][ip] = 0;
01886
                 help->h2o[ix][iy][ip] = 0;
01887
                 help->03[ix][iy][ip] = 0;
01888
                 wsum = 0;
                 for (ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1; ix2++) {
01889
01890
                  ix3 = ix2;
01891
                  if (ix3 < 0)
01892
                  ix3 += met->nx;
else if (ix3 >= met->nx)
01893
01894
                     ix3 -= met->nx;
01896
                   for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
                     iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
for (ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
    ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
    w = (float) (1.0 - fabs(ix - ix2) / ctl->met_sx)
    * (float) (1.0 - fabs(iy - iy2) / ctl->met_sy)
01897
01898
01899
01900
01901
                          * (float) (1.0 - fabs(ip - ip2) / ctl->met_sp);
01902
01903
                        help->ps[ix][iy] += w * met->ps[ix3][iy2];
                        help \rightarrow pt[ix][iy] += w * met \rightarrow pt[ix3][iy2];
01904
                        \label{eq:help-z} $$ $ help->z[ix][iy][ip] += w * met->z[ix3][iy2][ip2]; $$ $$
01905
                        help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
01906
                        help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
01907
01908
                       help \rightarrow v[ix][iy][ip] += w * met \rightarrow v[ix3][iy2][ip2];
01909
                        help \rightarrow w[ix][iy][ip] += w * met \rightarrow w[ix3][iy2][ip2];
                       help->pv[ix][iy][ip] += w * met->pv[ix3][iy2][ip2];
help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
01910
01911
                        help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
01912
01913
                        wsum += w;
01914
```

```
01915
01916
               help->ps[ix][iy] /= wsum;
01917
               help->pt[ix][iy] /= wsum;
               help->t[ix][iy][ip] /= wsum;
01918
               help->z[ix][iy][ip] /= wsum;
01919
               help->u[ix][iy][ip] /= wsum;
01920
               help->v[ix][iy][ip] /= wsum;
01921
01922
               help->w[ix][iy][ip] /= wsum;
01923
               help->pv[ix][iy][ip] /= wsum;
               help->h2o[ix][iy][ip] /= wsum;
01924
               help->o3[ix][iy][ip] /= wsum;
01925
01926
01927
          }
01928
01929
01930
        /* Downsampling... */
01931
        met->nx = 0;
        for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
01932
          met->lon[met->nx] = help->lon[ix];
01933
01934
          met->ny = 0;
01935
           for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
             met->lat[met->ny] = help->lat[iy];
01936
             met->ps[met->nx][met->ny] = help->ps[ix][iy];
met->pt[met->nx][met->ny] = help->pt[ix][iy];
01937
01938
             met->np = 0;
01939
             for (ip = 0; ip < help->np; ip += ctl->met_dp) {
01940
01941
               met->p[met->np] = help->p[ip];
01942
               met->z[met->nx][met->ny][met->np] = help->z[ix][iy][ip];
01943
               met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
01944
               met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
               met->v[met->nx] [met->ny] [met->np] = help->v[ix][iy][ip];
met->w[met->nx] [met->ny] [met->np] = help->w[ix][iy][ip];
01945
01946
01947
               met->pv[met->nx][met->ny][met->np] = help->pv[ix][iy][ip];
01948
               \verb|met->h2o[met->nx][met->ny][met->np]| = \verb|help->h2o[ix][iy][ip]|;
01949
               met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
01950
               met->np++;
01951
01952
             met->ny++;
01953
01954
          met->nx++;
01955
01956
         /* Free... */
01957
01958
        free(help);
01959 }
```

5.19.2.26 void read_met_tropo (ctl t * ctl, met t * met)

Calculate tropopause pressure.

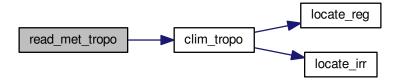
Definition at line 1963 of file libtrac.c.

```
01965
                        {
01966
01967
         gsl_interp_accel *acc;
01968
01969
        gsl_spline *spline;
01970
01971
        double p2[400], pv[400], pv2[400], t[400], t2[400], th[400], th2[400],
01972
          z[400], z2[400];
01973
01974
        int found, ix, iy, iz, iz2;
01975
01976
         /* Allocate... */
01977
         acc = gsl_interp_accel_alloc();
01978
         spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) met->np);
01979
01980
         /\star Get altitude and pressure profiles... \star/
        for (iz = 0; iz < met->np; iz++)
z[iz] = Z(met->p[iz]);
01981
01982
        for (iz = 0; iz <= 170; iz++) {
    z2[iz] = 4.5 + 0.1 * iz;
01983
01984
          p2[iz] = P(z2[iz]);
01985
01986
01987
01988
         /* Do not calculate tropopause... */
        if (ctl->met_tropo == 0)
  for (ix = 0; ix < met->nx; ix++)
01989
01990
01991
             for (iy = 0; iy < met->ny; iy++)
01992
               met->pt[ix][iy] = GSL_NAN;
```

```
01994
         /* Use tropopause climatology... */
01995
         else if (ctl->met_tropo == 1)
           for (ix = 0; ix < met->nx; ix++)
01996
              for (iy = 0; iy < met->ny; iy++)
  met->pt[ix][iy] = clim_tropo(met->time, met->lat[iy]);
01997
01998
01999
02000
         /* Use cold point... */
02001
         else if (ctl->met_tropo == 2) {
02002
            /* Loop over grid points... */
for (ix = 0; ix < met->nx; ix++)
02003
02004
              for (iy = 0; iy < met->ny; iy++) {
02005
02006
02007
                 /* Interpolate temperature profile... */
                 for (iz = 0; iz < met->np; iz++)
t[iz] = met->t[ix][iy][iz];
02008
02009
                 gsl_spline_init(spline, z, t, (size_t) met->np);
for (iz = 0; iz <= 170; iz++)</pre>
02010
02011
02012
                   t2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02013
02014
                 /* Find minimum... */
                 iz = (int) gsl_stats_min_index(t2, 1, 171);
if (iz <= 0 || iz >= 170)
  met->pt[ix][iy] = GSL_NAN;
02015
02016
02017
02018
02019
                   met->pt[ix][iy] = p2[iz];
02020
              }
02021
        }
02022
02023
         /* Use WMO definition... */
02024
         else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
02025
02026
            /\star Loop over grid points... \star/
02027
            for (ix = 0; ix < met->nx; ix++)
              for (iy = 0; iy < met->ny; iy++) {
02028
02029
                 /* Interpolate temperature profile... */
02031
                 for (iz = 0; iz < met->np; iz++)
02032
                   t[iz] = met->t[ix][iy][iz];
02033
                 gsl_spline_init(spline, z, t, (size_t) met->np);
                 for (iz = 0; iz <= 160; iz++)
  t2[iz] = gsl_spline_eval(spline, z2[iz], acc);</pre>
02034
02035
02036
02037
                 /* Find 1st tropopause... */
02038
                 met->pt[ix][iy] = GSL_NAN;
                 for (iz = 0; iz <= 140; iz++) {
  found = 1;
02039
02040
                   for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
02041
02042
                           / log(p2[iz2] / p2[iz]) > 2.0) {
02044
                         found = 0;
02045
                        break;
02046
02047
                    if (found) {
02048
                     if (iz > 0 && iz < 140)
                        met->pt[ix][iy] = p2[iz];
02050
                      break;
02051
02052
02053
                 /* Find 2nd tropopause... */
if (ctl->met_tropo == 4) {
  met->pt[ix][iy] = GSL_NAN;
02054
02055
02056
02057
                    for (; iz <= 140; iz++) {
02058
                      found = 1;
                      for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
02059
02060
                             / log(p2[iz2] / p2[iz]) < 3.0) {
02061
02062
                          found = 0;
02063
02064
02065
                      if (found)
02066
                        break;
02067
02068
                   for (; iz <= 140; iz++) {
                      found = 1;
02069
                      for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
  if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
     / log(p2[iz2] / p2[iz]) > 2.0) {
02070
02071
02072
02073
                          found = 0;
                           break;
02075
02076
                      if (found) {
02077
                       if (iz > 0 && iz < 140)
02078
                          met->pt[ix][iy] = p2[iz];
02079
                        break:
```

```
02081
                    }
02082
                 }
02083
02084
02085
02086
         /* Use dynamical tropopause... */
02087
          else if (ctl->met_tropo == 5) {
02088
            /* Loop over grid points... */
for (ix = 0; ix < met->nx; ix++)
02089
02090
02091
               for (iy = 0; iy < met->ny; iy++) {
02092
02093
                  /* Interpolate potential vorticity profile... */
                  for (iz = 0; iz < met->np; iz++)
   pv[iz] = met->pv[ix][iy][iz];
02094
02095
                  gsl_spline_init(spline, z, pv, (size_t) met->np);
for (iz = 0; iz <= 160; iz++)
   pv2[iz] = gsl_spline_eval(spline, z2[iz], acc);</pre>
02096
02097
02098
02099
02100
                  /* Interpolate potential temperature profile... */
                  for (iz = 0; iz < met->np; iz++)
  th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
02101
02102
                  gsl_spline_init(spline, z, th, (size_t) met->np); for (iz = 0; iz <= 160; iz++)
02103
02104
02105
                    th2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02106
02107
                  /\star Find dynamical tropopause 3.5 PVU + 380 K \star/
                  met->pt[ix][iy] = GSL_NAN;
02108
                  for (iz = 0; iz <= 160; iz++)
  if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
   if (iz > 0 && iz < 160)
02109
02110
02111
02112
                         met->pt[ix][iy] = p2[iz];
02113
                      break;
02114
                    }
               }
02115
02116
         }
02117
02118
02119
            ERRMSG("Cannot calculate tropopause!");
02120
         /* Free... */
gsl_spline_free(spline);
02121
02122
02123
          gsl_interp_accel_free(acc);
02124 }
```

Here is the call graph for this function:



5.19.2.27 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 2128 of file libtrac.c.

```
02135 {
02136
02137 FILE *in = NULL;
```

```
02138
02139
          char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
02140
            msg[2 * LEN], rvarname[LEN], rval[LEN];
02141
02142
         int contain = 0, i;
02143
02144
          /* Open file... */
         if (filename[strlen(filename) - 1] != '-')
if (!(in = fopen(filename, "r")))
02145
02146
02147
              ERRMSG("Cannot open file!");
02148
         /* Set full variable name... */
02149
02150
         if (arridx >= 0) {
          sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
02151
02152
02153
           sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
02154
02155
02156
02157
02158
         /* Read data... */
02159
         if (in != NULL)
         while (fgets(line, LEN, in))
if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
if (strcasecmp(rvarname, fullname1) == 0 ||
    strcasecmp(rvarname, fullname2) == 0) {
02160
02161
02162
02163
02164
                    contain = 1;
02165
                  break;
02166
                 }
         for (i = 1; i < argc - 1; i++)</pre>
02167
          if (strcasecmp(argv[i], fullname1) == 0 ||
02168
               strcasecmp(argv[i], fullname2) == 0) {
sprintf(rval, "%s", argv[i + 1]);
02169
02170
02171
              contain = 1;
02172
              break;
02173
02174
02175
         /* Close file... */
02176
         if (in != NULL)
02177
          fclose(in);
02178
02179
         /* Check for missing variables... */
02180
         if (!contain) {
          if (strlen(defvalue) > 0)
02181
              sprintf(rval, "%s", defvalue);
02183
            else
02184
              sprintf(msg, "Missing variable %s!\n", fullname1);
02185
              ERRMSG (msg);
02186
            }
02187
02188
02189
         /* Write info... */
02190
         printf("%s = %s\n", fullname1, rval);
02191
         /* Return values... */
if (value != NULL)
    sprintf(value, "%s", rval);
02192
02193
02195
         return atof(rval);
02196 }
```

5.19.2.28 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 2200 of file libtrac.c.

```
{
02210
       struct tm t0, t1;
02211
02212
       t0.tm vear = 100;
        t0.tm_mon = 0;
02213
02214
        t0.tm_mday = 1;
02215
        t0.tm\_hour = 0;
02216
        t0.tm_min = 0;
       t0.tm_sec = 0;
02217
02218
02219
       t1.tm vear = vear - 1900;
02220
       t1.tm_mon = mon - 1;
       t1.tm_mday = day;
```

```
02222    t1.tm_hour = hour;
02223    t1.tm_min = min;
02224    t1.tm_sec = sec;
02225
02226    *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
02227 }
```

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

Measure wall-clock time.

Definition at line 2231 of file libtrac.c.

```
02234
                     {
02235
02236
         static double starttime[NTIMER], runtime[NTIMER];
02237
         /* Check id... */
02238
         if (id < 0 || id >= NTIMER)
    ERRMSG("Too many timers!");
02239
02240
02241
02242
         /* Start timer... */
02243
         if (mode == 1) {
02244
          if (starttime[id] <= 0)</pre>
02245
             starttime[id] = omp_get_wtime();
02246
           else
02247
              ERRMSG("Timer already started!");
02248
02249
02250
         /* Stop timer... */
         else if (mode == 2) {
   if (starttime[id] > 0) {
02251
02252
             runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
02253
02254
              starttime[id] = -1;
02255
02256 }
02257
        /* Print timer... */
else if (mode == 3) {
  printf("%s = %.3f s\n", name, runtime[id]);
  runtime[id] = 0:
02258
02259
02260
02261
           runtime[id] = 0;
02262
         }
02263 }
```

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

Write atmospheric data.

Definition at line 2267 of file libtrac.c.

```
02271
                   {
02272
02273
        FILE *in. *out;
02274
02275
        char line[LEN];
02276
02277
        double r, t0, t1;
02278
02279
        int ip, iq, year, mon, day, hour, min, sec;
02280
02281
        /* Set time interval for output... */
02282
        t0 = t - 0.5 * ctl->dt_mod;
02283
        t1 = t + 0.5 * ctl->dt_mod;
02284
02285
        /* Write info... */
       printf("Write atmospheric data: %s\n", filename);
02286
02287
02288
        /* Write ASCII data...
02289
        if (ctl->atm_type == 0) {
02290
02291
          /* Check if gnuplot output is requested... */
if (ctl->atm_gpfile[0] != '-') {
02292
02293
            /* Create gnuplot pipe... */
```

```
if (!(out = popen("gnuplot", "w")))
02296
              ERRMSG("Cannot create pipe to gnuplot!");
02297
            /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
02298
02299
02300
            /* Set time string... */
            jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02302
02303
02304
                    year, mon, day, hour, min);
02305
            /* Dump gnuplot file to pipe... */
02306
02307
            if (!(in = fopen(ctl->atm_gpfile, "r")))
              ERRMSG("Cannot open file!");
02308
            while (fgets(line, LEN, in))
fprintf(out, "%s", line);
02309
02310
02311
            fclose(in);
02312
          }
02313
02314
          else {
02315
02316
            /* Create file... */
            if (!(out = fopen(filename, "w")))
02317
              ERRMSG("Cannot create file!");
02318
02319
02320
02321
          /* Write header... */
02322
          fprintf(out,
02323
                  "# $1 = time [s] \n"
                  "# $2 = altitude [km]\n"
02324
                  "# $3 = longitude [deg] \n" "# $4 = latitude [deg] \n");
02325
          02326
02327
02328
          fprintf(out, "\n");
02329
02330
          /* Write data... */
for (ip = 0; ip < atm->np; ip++) {
02331
02332
02333
02334
            /* Check time... */
02335
            02336
              continue;
02337
           /* Write output... */
fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
02338
02339
02340
                    atm->lon[ip], atm->lat[ip]);
            for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02341
02342
              fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02343
02344
02345
            fprintf(out, "\n");
02346
02347
02348
          /* Close file... */
02349
         fclose(out);
02350
02351
02352
        /* Write binary data... */
02353
        else if (ctl->atm_type == 1) {
02354
02355
          /* Create file... */
         if (!(out = fopen(filename, "w")))
02356
02357
            ERRMSG("Cannot create file!");
02358
02359
          /* Write data... */
02360
          FWRITE(&atm->np, int,
02361
                 1,
                 out);
02362
02363
          FWRITE(atm->time, double,
02364
                   (size_t) atm->np,
02365
                 out);
02366
          FWRITE(atm->p, double,
02367
                   (size_t) atm->np,
02368
                 out);
          FWRITE(atm->lon, double,
02369
02370
                  (size_t) atm->np,
02371
                 out);
02372
          FWRITE(atm->lat, double,
02373
                   (size_t) atm->np,
02374
                 out):
02375
          for (iq = 0; iq < ctl->nq; iq++)
           FWRITE(atm->q[iq], double,
02377
                     (size_t) atm->np,
02378
                   out);
02379
          /* Close file... */
02380
02381
          fclose(out);
```

```
02382 }
02383
02384 /* Error... */
02385 else
02386 ERRMSG("Atmospheric data type not supported!");
02387 }
```

Here is the call graph for this function:



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

Write CSI data.

Definition at line 2391 of file libtrac.c.

```
02395
02396
02397
         static FILE *in, *out;
02398
02399
         static char line[LEN];
02400
         static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
  rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
02401
02402
02403
02404
         static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
02405
02406
         /* Init... */
02407
         if (t == ctl->t_start) {
02408
02409
            /\star Check quantity index for mass... \star/
02410
           if (ctl->qnt_m < 0)
02411
              ERRMSG("Need quantity mass!");
02412
02413
            /\star Open observation data file... \star/
           printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
if (!(in = fopen(ctl->csi_obsfile, "r")))
02414
02415
02416
            ERRMSG("Cannot open file!");
02417
02418
            /* Create new file... */
           printf("Write CSI data: %s\n", filename);
if (!(out = fopen(filename, "w")))
02419
02420
              ERRMSG("Cannot create file!");
02421
02422
02423
            /* Write header... */
02424
            fprintf(out,
02425
                     "# $1 = time [s] \n"
                     "# $2 = number of hits (cx)\n"
02426
                     "# $3 = number of misses (cy)\n"
02427
                     "# $4 = number of false alarms (cz)\n"
02428
02429
                     "# $5 = number of observations (cx + cy) \n"
02430
                     "# $6 = number of forecasts (cx + cz) n"
                      "# \$7 = bias (forecasts/observations) [ \% ] \n"
02431
                     "# $8 = probability of detection (POD) [%%]\n" # $9 = false alarm rate (FAR) [%%]\n"
02432
02433
02434
                     "# $10 = critical success index (CSI) [%%]\n\n");
02435
02436
02437
         /\star Set time interval... \star/
         t0 = t - 0.5 * ctl->dt_mod;

t1 = t + 0.5 * ctl->dt_mod;
02438
02439
02440
02441
         /* Initialize grid cells... */
```

```
02442 #pragma omp parallel for default(shared) private(ix,iy,iz)
        for (ix = 0; ix < ctl->csi_nx; ix++)
02444
          for (iy = 0; iy < ctl->csi_ny; iy++)
           for (iz = 0; iz < ctl->csi_nz; iz++)
02445
02446
              modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
02447
        /* Read observation data... */
02449
        while (fgets(line, LEN, in)) {
02450
           /* Read data... */
02451
          if (sscanf(line, "%lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02452
02453
              5)
02454
            continue;
02455
02456
          /* Check time... */
02457
          if (rt < t0)
          continue;
if (rt > t1)
02458
02459
            break;
02460
02461
           /* Calculate indices... */
02462
02463
          ix = (int) ((rlon - ctl->csi_lon0))
          02464
02465
02466
          iz = (int) ((rz - ctl -> csi_z0))
02467
02468
                       / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02469
          /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
02470
02471
02472
              iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02473
            continue;
02474
02475
          /\star Get mean observation index... \star/
02476
          obsmean[ix][iy][iz] += robs;
02477
          obscount[ix][iy][iz]++;
02478
02480
        /* Analyze model data... */
02481 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
02482
        for (ip = 0; ip < atm->np; ip++) {
02483
02484
          /* Check time... */
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
02485
02486
            continue;
02487
02488
           /* Get indices... */
          ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
02489
                       / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02490
          iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
02491
                        / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02492
          02493
02494
02495
          /* Check indices... */
02496
          if (ix < 0 || ix >= ctl->csi_nx ||
02497
              iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02498
02499
            continue:
02500
02501
          /\star Get total mass in grid cell... \star/
          modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02502
02503
02504
02505
        /* Analyze all grid cells... */
02506 #pragma omp parallel for default(shared) private(ix,iy,iz,dlon,dlat,lat,area)
02507
       for (ix = 0; ix < ctl->csi_nx; ix++)
02508
          for (iy = 0; iy < ctl->csi_ny; iy++)
  for (iz = 0; iz < ctl->csi_nz; iz++) {
02509
02510
               /* Calculate mean observation index... */
02512
               if (obscount[ix][iy][iz] > 0)
02513
                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
02514
               /\star Calculate column density... \star/
02515
               if (modmean[ix][iy][iz] > 0) {
  dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
02516
02517
                 dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
02518
                lat = (ctr >csi_lat0 + dlat * (iy + 0.5);
area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
  * cos(lat * M_PI / 180.);
modmean[ix][iy][iz] /= (le6 * area);
02519
02520
02521
02522
02524
02525
               /* Calculate CSI... */
02526
               if (obscount[ix][iy][iz] > 0) {
                if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
    modmean[ix][iy][iz] >= ctl->csi_modmin)
02527
02528
```

```
cx++;
02530
                    else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02531
                                 modmean[ix][iy][iz] < ctl->csi_modmin)
02532
                      cv++;
02533
                    else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
    modmean[ix][iy][iz] >= ctl->csi_modmin)
02534
02535
                       cz++;
02536
02537
               }
02538
          /* Write output... */
if (fmod(t, ctl->csi_dt_out) == 0) {
02539
02540
02541
            /* Write... */ fprintf(out, \ "% .2f %d %d %d %d %d %g %g %g %g %g \n", \\
02542
02543
                       (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,

(cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,

(cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
02544
02545
02546
02547
02548
                       (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
02549
02550
             /\star Set counters to zero... \star/
02551
            cx = cy = cz = 0;
02552
02553
02554
         /* Close file... */
          if (t == ctl->t_stop)
02555
02556
            fclose(out);
02557 }
```

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

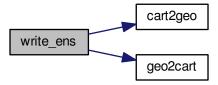
Write ensemble data.

Definition at line 2561 of file libtrac.c.

```
02566
02567
        static FILE *out;
02568
        static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
02569
          t0, t1, x[NENS][3], xm[3];
02571
02572
        static int ip, iq;
02573
02574
        static size_t i, n;
02575
02576
        /* Init... */
02577
        if (t == ctl->t_start) {
02578
02579
          /* Check quantities... */
02580
          if (ctl->qnt_ens < 0)
   ERRMSG("Missing ensemble IDs!");</pre>
02581
02582
02583
          /\star Create new file... \star/
          printf("Write ensemble data: %s\n", filename);
if (!(out = fopen(filename, "w")))
02584
02585
            ERRMSG("Cannot create file!");
02586
02587
02588
          /* Write header... */
          fprintf(out,
02590
                   "# $1 = time [s] \n"
02591
                   "# $2 = altitude [km] \n"
                   "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02592
          for (iq = 0; iq < ctl->nq; iq++)
fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
02593
02594
                     ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02595
          02596
02597
02598
02599
02600
02601
02602
        /* Set time interval... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
02603
02604
02605
02606
        /* Init...
02607
        ens = GSL_NAN;
02608
        n = 0;
```

```
02609
02610
         /* Loop over air parcels... */
02611
         for (ip = 0; ip < atm->np; ip++) {
02612
02613
           /* Check time... */
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
02614
02615
             continue;
02616
02617
           /* Check ensemble id... */
02618
           if (atm->q[ctl->qnt_ens][ip] != ens) {
02619
02620
             /* Write results... */
02621
             if (n > 0) {
02622
02623
                /∗ Get mean position...
               for (i = 0; i < n; i++) {
    xm[0] += x[i][0] / (double) n;
    xm[1] += x[i][1] / (double) n;</pre>
02624
02625
02626
02627
                  xm[2] += x[i][2] / (double) n;
02628
02629
02630
                cart2geo(xm, &dummy, &lon, &lat);
                02631
02632
                         lat):
02633
02634
                /* Get quantity statistics... */
                for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02635
02636
02637
                  fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02638
                for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02639
02640
02641
                  fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02642
02643
                fprintf(out, " lu\n", n);
02644
02645
02646
              /* Init new ensemble... */
02647
             ens = atm->q[ctl->qnt_ens][ip];
02648
             n = 0;
02649
02650
           /* Save data... */
02651
02652
           p[n] = atm -> p[ip];
           geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
02653
02654
           for (iq = 0; iq < ctl->nq; iq++)
           q[iq][n] = atm->q[iq][ip];
if ((++n) >= NENS)
02655
02656
             ERRMSG("Too many data points!");
02657
02658
02659
02660
         /* Write results... */
02661
         if (n > 0) {
02662
02663
           /* Get mean position... */
           xm[0] = xm[1] = xm[2] = 0;
for (i = 0; i < n; i++) {
02664
02665
             xm[0] += x[i][0] / (double) n;
xm[1] += x[i][1] / (double) n;
xm[2] += x[i][2] / (double) n;
02666
02667
02668
02669
           cart2geo(xm, &dummy, &lon, &lat);
fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
02670
02671
02672
02673
           /* Get quantity statistics... */
           for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02674
02675
02676
             fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02677
           for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02679
02680
              fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02681
           fprintf(out, " lu\n", n);
02682
02683
02684
02685
         /* Close file... */
02686
         if (t == ctl->t_stop)
02687
           fclose(out);
02688 }
```

Here is the call graph for this function:



5.19.2.33 void write grid (const char * filename, ctl t * ctl, met t * met0, met t * met1, atm t * atm, double t)

Write gridded data.

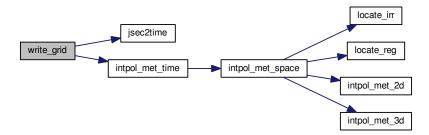
Definition at line 2692 of file libtrac.c.

```
02698
                      {
02700
         FILE *in, *out;
02701
02702
         char line[LEN];
02703
02704
         static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
02705
           area, rho_air, press, temp, cd, vmr, t0, t1, r;
02706
02707
         static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
02708
02709
         /* Check dimensions... */
         if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
02710
02711
           ERRMSG("Grid dimensions too large!");
02712
02713
         /\star Check quantity index for mass... \star/
02714
         if (ctl->qnt_m < 0)
02715
           ERRMSG("Need quantity mass!");
02716
02717
         /* Set time interval for output... */
02718
         t0 = t - 0.5 * ctl->dt_mod;
         t1 = t + 0.5 * ctl->dt_mod;
02719
02720
         /* Set grid box size... */
dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
02721
02722
02723
02724
         dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
02725
          /* Initialize grid... */
02726
02727 #pragma omp parallel for default(shared) private(ix,iy,iz)
         for (ix = 0; ix < ctl->grid_nx; ix++)
02728
           for (iy = 0; iy < ctl->grid_ny; iy++)
for (iz = 0; iz < ctl->grid_nz; iz++)
02729
02730
02731
                mass[ix][iy][iz] = 0;
02732
02733 /* Average data... */
02734 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
02735 for (ip = 0; ip < atm->np; ip++)
02736
           if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {</pre>
02737
02738
              /* Get index... */
              ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
02739
02740
02741
02742
02743
              /* Check indices... */
02744
              if (ix < 0 || ix >= ctl->grid_nx ||
                   iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
02745
02746
                 continue;
02747
02748
              /* Add mass... */
02749
              mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
```

```
02750
02751
02752
         /\star Check if gnuplot output is requested... \star/
         if (ctl->grid_gpfile[0] != '-') {
02753
02754
02755
           /* Write info... */
02756
           printf("Plot grid data: %s.png\n", filename);
02757
02758
            /* Create gnuplot pipe... */
           if (!(out = popen("gnuplot", "w")))
02759
             ERRMSG("Cannot create pipe to gnuplot!");
02760
02761
02762
            /* Set plot filename...
           fprintf(out, "set out \"%s.png\"\n", filename);
02763
02764
02765
           /* Set time string... */
           jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02766
02767
                    year, mon, day, hour, min);
02769
02770
            /* Dump gnuplot file to pipe... */
           if (!(in = fopen(ctl->grid_gpfile, "r")))
    ERRMSG("Cannot open file!");
02771
02772
02773
           while (fgets(line, LEN, in))
fprintf(out, "%s", line);
02774
02775
           fclose(in);
02776
02777
02778
         else {
02779
02780
           /* Write info... */
02781
           printf("Write grid data: %s\n", filename);
02782
           /* Create file... */
02783
          if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");
02784
02785
02786
02787
02788
         /* Write header... */
02789
         fprintf(out,
02790
                   "# $1 = time [s] \n"
                  "# $2 = altitude [km] \n"
02791
                   "# $3 = longitude [deg]\n"
02792
                  "# $4 = latitude [deg]\n"
02793
02794
                  "# $5 = surface area [km^2]\n"
02795
                  "# $6 = layer width [km] \n"
                  "# $7 = temperature [K]\n"
"# $8 = column density [kg/m^2]\n"
02796
02797
                  "# $9 = volume mixing ratio [1] n n;
02798
02799
        /* Write data... */
02801
        for (ix = 0; ix < ctl->grid_nx; ix++) {
02802
          if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
           fprintf(out, "\n");
for (iy = 0; iy < ctl->grid_ny; iy++) {
   if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
02803
02804
02805
                fprintf(out, "\n");
02806
02807
              for (iz = 0; iz < ctl->grid_nz; iz++)
02808
                if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
02809
                  /* Set coordinates... */
02810
                  z = ctl->grid_z0 + dz * (iz + 0.5);
lon = ctl->grid_lon0 + dlon * (ix + 0.5);
02811
02812
02813
                  lat = ctl->grid_lat0 + dlat * (iy + 0.5);
02814
02815
                  /\star Get pressure and temperature... \star/
02816
                  press = P(z);
                  intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
02817
                                     NULL, &temp, NULL, NULL, NULL, NULL, NULL, NULL);
02818
02820
                  /* Calculate surface area... */
                  area = dlat * dlon * SQR(RE * M_PI / 180.)
* cos(lat * M_PI / 180.);
02821
02822
02823
                  /* Calculate column density... */
02824
                  cd = mass[ix][iy][iz] / (1e6 * area);
02825
02826
02827
                  /\star Calculate volume mixing ratio... \star/
                  rho_air = 100. * press / (RA * temp);
vmr = MA / ctl->molmass * mass[ix][iy][iz]
    / (rho_air * 1e6 * area * 1e3 * dz);
02828
02829
02830
02831
                  02832
02833
02834
02835
                }
02836
           }
```

```
02837 }
02838
02839 /* Close file... */
02840 fclose(out);
02841 }
```

Here is the call graph for this function:



5.19.2.34 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 2845 of file libtrac.c.

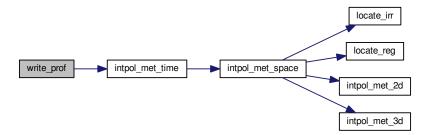
```
02851
                       {
02852
02853
          static FILE *in, *out;
02854
02855
         static char line[LEN];
02856
02857
          \verb|static double mass[GX][GY][GZ]|, obsmean[GX][GY]|, obsmean2[GX][GY]|, rt, rz|, \\
02858
           rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp,
02859
            rho_air, vmr, h2o, o3;
02860
02861
          static int obscount[GX][GY], ip, ix, iy, iz, okay;
02862
02863
          /* Init... */
02864
          if (t == ctl->t_start) {
02865
02866
            /* Check quantity index for mass... */
02867
            if (ctl->qnt_m < 0)
02868
               ERRMSG("Need quantity mass!");
02869
            /* Check dimensions... */
if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
    ERRMSG("Grid dimensions too large!");
02870
02871
02872
02873
02874
             /* Open observation data file... */
02875
            printf("Read profile observation data: %s\n", ctl->prof_obsfile);
            if (!(in = fopen(ctl->prof_obsfile, "r")))
    ERRMSG("Cannot open file!");
02876
02877
02878
            /* Create new output file... */
printf("Write profile data: %s\n", filename);
if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");
02879
02880
02881
02882
02883
             /* Write header... */
02884
02885
            fprintf(out,
02886
                       "# $1 = time [s] \n"
02887
                       "# $2 = altitude [km] \n"
                      "# $3 = longitude [deg]\n"
"# $4 = latitude [deg]\n"
"# $5 = pressure [hPa]\n"
02888
02889
02890
02891
                       "# $6 = temperature [K] \n"
02892
                       "# $7 = volume mixing ratio [1]\n"
```

```
"# $8 = H20 volume mixing ratio [1]\n"
02894
                      "# $9 = 03 volume mixing ratio [1]\n"
02895
                      "# $10 = observed BT index (mean) [K] \n"
                      "# $11 = observed BT index (sigma) [K] n");
02896
02897
            /* Set grid box size... */
02898
            dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
           dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
02900
02901
02902
02903
         /* Set time interval... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
02904
02905
02906
02907
         /* Initialize... */
02908
02909 #pragma omp parallel for default(shared) private(ix,iy,iz)
02910 for (ix = 0; ix < ctl->prof_nx; ix++)
          for (iy = 0; iy < ctl->prof_ny; iy++) {
02911
02912
              obsmean[ix][iy] = 0;
02913
              obsmean2[ix][iy] = 0;
02914
              obscount[ix][iy] = 0;
              for (iz = 0; iz < ctl->prof_nz; iz++)
02915
02916
                mass[ix][iy][iz] = 0;
02917
02918
         /\star Read observation data... \star/
02919
02920
         while (fgets(line, LEN, in)) {
02921
            /* Read data... */
if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02922
02923
02924
                 5)
02925
              continue;
02926
02927
            /* Check time... */
            if (rt < t0)</pre>
02928
02929
              continue;
            if (rt > t1)
02930
02931
              break;
02932
           /* Calculate indices... */
ix = (int) ((rlon - ctl->prof_lon0) / dlon);
iy = (int) ((rlat - ctl->prof_lat0) / dlat);
02933
02934
02935
02936
02937
            /* Check indices... */
02938
            if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
02939
             continue;
02940
            /* Get mean observation index... */
02941
            obsmean[ix][iy] += robs;
obsmean2[ix][iy] += SQR(robs);
02942
02943
02944
            obscount[ix][iy]++;
02945
02946
02947
         /\star Analyze model data... \star/
02948 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
02949 for (ip = 0; ip < atm->np; ip++) {
02950
            /* Check time... */
02951
02952
            if (atm->time[ip] < t0 || atm->time[ip] > t1)
02953
              continue;
02954
02955
            /* Get indices... */
           ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
02956
02957
            iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
02958
02959
02960
            /* Check indices... */
            if (ix < 0 || ix >= ctl->prof_nx ||
02961
                iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
02962
02963
02964
           /* Get total mass in grid cell... */ mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02965
02966
02967
02968
02969
          /* Extract profiles... */
         for (ix = 0; ix < ctl->prof_nx; ix++)
  for (iy = 0; iy < ctl->prof_ny; iy++)
02970
02971
              if (obscount[ix][iy] > 0) {
02972
02973
02974
                 /* Check profile... */
02975
                 okay = 0;
                 for (iz = 0; iz < ctl->prof_nz; iz++)
02976
02977
                  if (mass[ix][iy][iz] > 0) {
02978
                     okay = 1;
break;
02979
```

```
02980
02981
               if (!okay)
02982
                 continue;
02983
               /* Write output... */
02984
               fprintf(out, "\n");
02985
02986
02987
                /* Loop over altitudes... */
02988
               for (iz = 0; iz < ctl->prof_nz; iz++) {
02989
02990
                 /* Set coordinates... */
                 z = ctl->prof_z0 + dz * (iz + 0.5);
lon = ctl->prof_lon0 + dlon * (ix + 0.5);
lat = ctl->prof_lat0 + dlat * (iy + 0.5);
02991
02992
02993
02994
                 /* Get pressure and temperature... */ press = P(z);
02995
02996
                 02997
02998
02999
                 /* Calculate surface area... */
area = dlat * dlon * SQR(M_PI * RE / 180.)
    * cos(lat * M_PI / 180.);
03000
03001
03002
03003
03004
                 /* Calculate volume mixing ratio... */
03005
                 rho_air = 100. * press / (RA * temp);

vmr = MA / ctl->molmass * mass[ix][iy][iz]
03006
03007
                   / (rho_air * area * dz * 1e9);
03008
                 /* Write output... */
03009
                 03010
03011
03012
03013
                          sqrt(obsmean2[ix][iy] / obscount[ix][iy]
                                - SQR(obsmean[ix][iy] / obscount[ix][iy])));
03014
03015
03016
03017
03018
        /* Close file... */
03019
        if (t == ctl->t_stop)
03020
           fclose(out);
03021 }
```

Here is the call graph for this function:



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

Write station data.

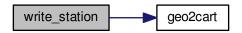
Definition at line 3025 of file libtrac.c.

```
03029 {
03030
03031 static FILE *out;
03032
03033 static double rmax2, t0, t1, x0[3], x1[3];
```

```
03034
03035
         static int ip, iq;
03036
03037
         /* Init... */
03038
         if (t == ctl->t start) {
03039
03040
            /* Write info... */
03041
           printf("Write station data: %s\n", filename);
03042
           /* Create new file... */
if (!(out = fopen(filename, "w")))
03043
03044
             ERRMSG("Cannot create file!");
03045
03046
03047
            /* Write header... */
03048
           fprintf(out,
03049
                     "# $1 = time [s] \n"
                     "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03050
03051
           for (iq = 0; iq < ctl->nq; iq++)

fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
03052
03053
                       ctl->qnt_name[iq], ctl->qnt_unit[iq]);
03054
03055
           fprintf(out, "\n");
03056
           /* Set geolocation and search radius... */ geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
03057
03058
03059
           rmax2 = SQR(ctl->stat_r);
03060
03061
         /* Set time interval for output... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
03062
03063
03064
03065
03066
         /* Loop over air parcels... */
03067
         for (ip = 0; ip < atm->np; ip++) {
03068
03069
            /* Check time... */
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
03070
03071
             continue;
03072
03073
            /* Check station flag... */
03074
           if (ctl->qnt_stat >= 0)
03075
            if (atm->q[ctl->qnt_stat][ip])
03076
                continue:
03077
03078
            /\star Get Cartesian coordinates... \star/
03079
           geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
03080
03081
            /\star Check horizontal distance... \star/
           if (DIST2(x0, x1) > rmax2)
03082
03083
             continue:
03084
03085
            /* Set station flag... */
03086
            if (ctl->qnt_stat >= 0)
03087
             atm->q[ctl->qnt_stat][ip] = 1;
03088
03089
           /* Write data... */
fprintf(out, "%.2f %g %g %g",
03090
03091
                    atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
            for (iq = 0; iq < ctl->nq; iq++) {
   fprintf(out, " ");
03092
03093
              fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
03094
03095
03096
           fprintf(out, "\n");
03097
03098
03099
         /* Close file... */
         if (t == ctl->t_stop)
03100
03101
           fclose(out);
03102 }
```

Here is the call graph for this function:



5.20 libtrac.c

```
00001 /*
         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
         the Free Software Foundation, either version 3 of the License, or
00006
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
         You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
00016
00017
         Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00028
00029 void cart2geo(
00030
         double *x,
00031
         double *z.
00032
         double *lon,
00033
         double *lat)
00034
00035
         double radius;
00036
00037
         radius = NORM(x);
00038
         *lat = asin(x[2] / radius) * 180 / M_PI;
00039
         *lon = atan2(x[1], x[0]) * 180 / M_PI;
00040
         *z = radius - RE;
00041 }
00042
00044
00045 double clim hno3(
         double t,
00046
00047
         double lat,
00048
         double p) {
00049
         static double secs[12] = { 1209600.00, 3888000.00, 6393600.00,
00050
            9072000.00, 11664000.00, 14342400.00,
00052
            16934400.00, 19612800.00, 22291200.00,
00053
            24883200.00, 27561600.00, 30153600.00
00054
00055
00056
         static double lats[18] = { -85, -75, -65, -55, -45, -35, -25, -15, -5,
            5, 15, 25, 35, 45, 55, 65, 75, 85
00057
00058
00059
         static double ps[10] = { 4.64159, 6.81292, 10, 14.678, 21.5443,
00060
00061
           31.6228, 46.4159, 68.1292, 100, 146.78
00062
00063
00064
          static double hno3[12][18][10] = {
00065
           {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
             {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57}, {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54}, {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00066
00067
00068
00069
             \{0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709\},\
              \{0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37\},
00071
              {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00072
              {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
             {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181}, {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104}, {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985}, {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00073
00074
00075
              {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
00077
00078
              {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
              {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00079
             {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97}, {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14}, {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
08000
00081
00082
            {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00083
             {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42}, {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33}, {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05}, {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661}, {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
00084
00085
00086
00087
00088
             {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
```

5.20 libtrac.c 99

```
{0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
                {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.1671, {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
00091
00092
00093
                {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00094
                {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191}, {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
00095
                {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
                {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65,
00097
00008
                {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
                {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07}, {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
00099
00100
              {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00101
                \{0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52\},
00102
                {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
00103
                {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98}, {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642}, {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33}
00104
00105
00106
                {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174}, {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
00107
00109
                {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
                \{0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121\},
00110
00111
                {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
                {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194}, {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1}, {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
00112
00113
00114
                {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82}, {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
00116
00117
                {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
              {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}}, {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75}, {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00118
00119
00120
00121
                {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
                {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09}
00122
00123
                {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727}
                {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409}, {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
00124
00125
                {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12}, {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
00126
00128
                {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
                {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
00129
                {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286}, {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
00130
00131
                {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96}, {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
00132
00133
                {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
00135
                {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
00136
                {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
              {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63}, {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
00137
00138
                {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63}, {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
00139
                {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88}, {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
00141
00142
                {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229}, {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
00143
00144
                \{0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126\},
00145
                \{0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183\},
                {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
00147
                {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
00148
                {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964}, {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83}, {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25}, {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
00149
00150
00151
                {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52}, {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}},
00153
00154
              {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26}, {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05}, {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
00155
00156
00157
                {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
00158
                {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
                {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
00160
00161
                {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217}
00162
                {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
                {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136}, {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194}, {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
00163
00164
                \{0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302\},
00166
00167
                {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
                {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41}, {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
00168
00169
                {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
00170
                {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
                \{0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91\}\},
00172
00173
               {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
00174
                {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
                {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08}, {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
00175
```

```
{1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
              {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656}, {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
00178
00179
              {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
00180
              {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12}
00181
              {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
00182
              {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25}
              {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
00184
00185
              {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
00186
              {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
              {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
00187
00188
              \{0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56\},
00189
              {0.735, 1.51, 2.96, 4.84, 5.92,
                                                        7.77,
                                                                7.2, 5.54, 2.56, 1.61},
              \{0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62\}\},
00190
00191
             {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
              {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73}, {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
00192
00193
              {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
00194
              {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
              {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
00196
              {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
00197
00198
              {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
              {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107}, {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185}, {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
00199
00200
00201
              {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232
00203
              {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
00204
              {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
00205
              {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
              {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
00206
              {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
00207
00208
              \{0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55\}\},
             {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
00209
00210
              {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74}
              {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09}, {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83}, {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22}, {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646}, {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
00211
00212
00213
00215
              {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
00216
00217
              {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
              {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147}, 
{1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197}, 
{1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
00218
00219
00220
              \{0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303\},
00221
00222
              {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714}
              {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1}, {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41}, {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56}, {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
00223
00224
00225
00226
             {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
              {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
00228
00229
              {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
              {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56}, {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
00230
00231
00232
              {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
              \{0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21\},
              {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
00234
              {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
00235
              {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146}, {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
00236
00237
              {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14}, {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
00238
              \{0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802\},\
00240
00241
              {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
              {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
00242
00243
              {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
00244
              {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}}
00245
             {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
              {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
              \{0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75\},\
00247
00248
              {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
00249
              \{0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955\},
00250
              \{0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61\},
              {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
00251
              {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
00252
              \{0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121\},
00253
00254
              {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
              {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146}, {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172}, {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448}, {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
00255
00256
00257
              {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
00259
00260
              {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
             {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97}, {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}}, {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
00261
00262
00263
```

5.20 libtrac.c 101

```
{0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
              {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65}, {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
00265
00266
              {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
00267
              {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488}, {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
00268
00269
              (0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198), (0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173),
00271
00272
              {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
              {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133}, {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189}, {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
00273
00274
00275
00276
              {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
              {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
00277
00278
              {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
              {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35}, {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
00279
00280
00281
          };
00283
          double aux00, aux01, aux10, aux11, sec;
00284
00285
          int ilat, ip, isec;
00286
         /* Get seconds since begin of year... */
sec = fmod(t, 365.25 * 86400.);
00287
00288
00290
          /* Get indices... */
         isec = locate_irr(secs, 12, sec);
ilat = locate_reg(lats, 18, lat);
00291
00292
00293
          ip = locate_irr(ps, 10, p);
00294
00295
          /* Interpolate...
00296
          aux00 = LIN(ps[ip], hno3[isec][ilat][ip],
00297
                         ps[ip + 1], hno3[isec][ilat][ip + 1], p);
00298
          aux01 = LIN(ps[ip], hno3[isec][ilat + 1][ip],
                         ps[ip + 1], hno3[isec][ilat + 1][ip + 1], p);
00299
00300
          aux10 = LIN(ps[ip], hno3[isec + 1][ilat][ip],
                         ps[ip + 1], hno3[isec + 1][ilat][ip + 1], p);
00302
          aux11 = LIN(ps[ip], hno3[isec + 1][ilat + 1][ip],
00303
                         ps[ip + 1], hno3[isec + 1][ilat + 1][ip + 1], p);
00304
          aux00 = LIN(lats[ilat], aux00, lats[ilat + 1], aux01, lat);
          aux11 = LIN(lats[ilat], aux10, lats[ilat + 1], aux11, lat);
00305
00306
          return LIN(secs[isec], aux00, secs[isec + 1], aux11, sec);
00307 }
00310
00311 double clim_tropo(
         double t,
00312
00313
         double lat) {
00314
00315
          static double doys[12]
00316
          = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00317
          static double lats[73]
00318
            = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5, -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5, -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
00319
00321
            -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5, 15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
00322
00323
             45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
00324
00325
            75, 77.5, 80, 82.5, 85, 87.5, 90
00326
00327
00328
          static double tps[12][73]
00329
            = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
00330
                   297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4, 175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
00331
                   99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54, 98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5,
00332
00333
                   152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275, 277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
00334
00335
          275.3, 275.6, 275.4, 274.1, 273.5}, {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7, 300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
00336
00337
00338
           150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
            98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
00340
00341
           98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193, 220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
00342
00343
           284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
           287.5, 286.2, 285.8},
00344
00345
          {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
           297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
00346
00347
           161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
00348
           100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
           99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8, 186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
00349
00350
```

```
279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
          304.3, 304.9, 306, 306.6, 306.2, 306}, {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
00352
00353
00354
           290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
00355
           195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7, 102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
00356
            99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
            148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
00358
00359
            263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
          315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1}, {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8, 260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
00360
00361
00362
            205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7,
00363
                                                                                    104.1,
                                                                                              102.7.
            101.8, 101.4, 101.1, 101, 101, 101.1, 101.2, 101.5, 101.9,
00364
00365
            102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
00366
            165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
00367
           273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
            325.3, 325.8, 325.8},
00368
           {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
           222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2, 228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1, 105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
00370
00371
00372
           106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6, 127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245, 251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
00373
00374
00375
            308.5, 312.2, 313.1, 313.3},
           {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
00377
00378
           187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
           235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1, 110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7, 111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
00379
00380
00381
00382
            117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
            224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5
00383
00384
            275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
          {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4, 185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3, 233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
00385
00386
00387
            110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
00389
            112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
           120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
00390
00391
           230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7
           278.2, 282.6, 287.4, 290.9, 292.5, 293},
00392
          {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3, 183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
00393
00394
            243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
00395
00396
           114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4,
                                                                                       110.5
00397
           110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
           114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9, 203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
00398
00399
           276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
00400
          {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
           215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
00402
00403
           237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
           111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7, 106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
00404
00405
           112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7, 206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
00406
            279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
00408
           305.1},
00409
           {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
00410
00411
           253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
           223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9, 108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
00412
00413
            102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
00414
00415
           109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
00416
           241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
00417
            286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6}
          284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
00418
           175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1, 100.4, 99.96, 99.6, 99.37, 99.32, 99.31, 99.46, 99.77, 100.2 100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
00421
00422
           186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278, 280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
00423
00424
           281.7, 281.1, 281.2}
00425
00426
00427
00428
          double doy, p0, p1;
00429
00430
          int imon, ilat:
00431
00432
          /* Get day of year... */
          doy = fmod(t / 86400., 365.25);
00433
00434
          while (doy < 0)
00435
            doy += 365.25;
00436
00437
          /* Get indices... */
```

5.20 libtrac.c 103

```
ilat = locate_reg(lats, 73, lat);
00439
       imon = locate_irr(doys, 12, doy);
00440
00441
       /* Interpolate... */
      00442
00443
00445
00446
       return LIN(doys[imon], p0, doys[imon + 1], p1, doy);
00447 }
00448
00450
00451 void day2doy(
00452
       int year,
00453
       int mon,
00454
       int day.
00455
       int *doy)
00456
       int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00458
       int d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00459
       /* Get day of year... */
if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
*doy = d01[mon - 1] + day - 1;
00460
00461
00462
00463
00464
         *doy = d0 [mon - 1] + day - 1;
00465 }
00466
00468
00469 void doy2day(
00470
      int year,
00471
       int doy,
00472
       int *mon,
00473
       int *day) {
00474
       int d0[12] = \{ 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 \};
00476
       int d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00477
00478
       /* Get month and day... */
if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
  for (i = 11; i >= 0; i--)
   if (d01[i] <= doy)</pre>
00479
00480
00481
00483
            break;
00484
         *mon = i + 1;
00485
         *day = doy - d01[i] + 1;
       } else {
00486
        for (i = 11; i >= 0; i--)
00487
         if (d0[i] <= doy)
00488
00489
            break;
00490
         *mon = i + 1;
00491
        *day = doy - d0[i] + 1;
00492
00493 }
00496
00497 void geo2cart(
00498
       double z,
00499
       double lon,
00500
       double lat,
00501
       double *x) {
00502
00503
       double radius;
00504
00505
       radius = z + RE:
       x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
x[2] = radius * sin(lat / 180 * M_PI);
00506
00508
00509 }
00510
00512
00513 void get_met(
00514
       ctl_t * ctl,
00515
       char *metbase,
00516
       double t,
      met_t ** met0,
met_t ** met1) {
00517
00518
00519
00520
       static int init, ip, ix, iy;
00521
00522
       met_t *mets;
00523
00524
       char filename[LEN]:
```

```
00525
00526
        /* Init... */
00527
        if (t == ctl->t_start || !init) {
         init = 1;
00528
00529
         get_met_help(t, -1, metbase, ctl->dt_met, filename);
00530
         read_met(ctl, filename, *met0);
00531
00532
00533
          get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
     dt_met, filename);
00534
         read_met(ctl, filename, *metl);
00535
00536
00537
        /* Read new data for forward trajectories... */
00538
        if (t > (*met1)->time && ctl->direction == 1) {
        mets = *met1;
*met1 = *met0;
00539
00540
         *met0 = mets;
00541
00542
         get_met_help(t, 1, metbase, ctl->dt_met, filename);
00543
         read_met(ctl, filename, *met1);
00544
00545
00546
        /* Read new data for backward trajectories... */
00547
        if (t < (*met0) -> time && ctl->direction == -1) {
00548
         mets = *met1;
         *met1 = *met0;
00549
00550
          *met0 = mets;
00551
          get_met_help(t, -1, metbase, ctl->dt_met, filename);
00552
          read_met(ctl, filename, *met0);
00553
00554
00555
        /* Check that grids are consistent... */
00556
        if ((*met0)->nx != (*met1)->nx
00557
            |\ |\ (*met0) -> ny \ != \ (*met1) -> ny \ |\ (*met0) -> np \ != \ (*met1) -> np)
        ERRMSG("Meteo grid dimensions do not match!");
for (ix = 0; ix < (*met0)->nx; ix++)
  if ((*met0)->lon[ix] != (*met1)->lon[ix])
00558
00559
00560
           ERRMSG("Meteo grid longitudes do not match!");
00561
00562
        for (iy = 0; iy < (*met0) ->ny; iy++)
00563
            ((*met0)->lat[iy] != (*met1)->lat[iy])
00564
           ERRMSG("Meteo grid latitudes do not match!");
00565
        for (ip = 0; ip < (*met0)->np; ip++)
if ((*met0)->p[ip] != (*met1)->p[i]
00566
            ((*met0) - p[ip] != (*met1) - p[ip])
00567
            ERRMSG("Meteo grid pressure levels do not match!");
00568 }
00569
00571
00572 void get_met_help(
00573
       double t.
        int direct,
00575
        char *metbase,
00576
       double dt_met,
00577
       char *filename) {
00578
00579
       double t6, r;
00580
00581
        int year, mon, day, hour, min, sec;
00582
00583
        /\star Round time to fixed intervals... \star/
00584
        if (direct == -1)
         t6 = floor(t / dt_met) * dt_met;
00585
00586
        else
00587
         t6 = ceil(t / dt_met) * dt_met;
00588
00589
        /* Decode time... */
00590
        jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00591
        /* Set filename... */
00592
        sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", metbase, year, mon, day, hour);
00593
00594 }
00595
00597
00598 void intpol_met_2d(
00599
       double array[EX][EY],
        int ix,
00600
00601
        int iy,
00602
       double wx,
00603
       double wy,
00604
       double *var) {
00605
       double aux00, aux01, aux10, aux11;
00606
00607
       /* Set variables... */
00608
00609
       aux00 = array[ix][iy];
       aux01 = array[ix][iy + 1];
00610
```

5.20 libtrac.c 105

```
aux10 = array[ix + 1][iy];
        aux11 = array[ix + 1][iy + 1];
00612
00613
00614
        /* Interpolate horizontally... */
00615
        aux00 = wy * (aux00 - aux01) + aux01;

aux11 = wy * (aux10 - aux11) + aux11;
00616
00617
        *var = wx * (aux00 - aux11) + aux11;
00618 }
00619
00621
00622 void intpol_met_3d(
00623
        float array[EX][EY][EP],
00624
        int ip,
00625
        int ix,
00626
        int iy,
00627
        double wp,
00628
        double wx,
00629
        double wy,
00630
        double *var) {
00631
00632
        double aux00, aux01, aux10, aux11;
00633
        /* Interpolate vertically... */
aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00634
00635
          + array[ix][iy][ip + 1];
00637
        aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
        + array[ix][iy + 1][ip + 1];
aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
00638
00639
        + array[ix + 1][iy][ip + 1];
aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
+ array[ix + 1][iy + 1][ip + 1];
00640
00641
00642
00643
00644
        /* Interpolate horizontally... */
        aux00 = wy * (aux00 - aux01) + aux01;

aux11 = wy * (aux10 - aux11) + aux11;
00645
00646
00647
        *var = wx * (aux00 - aux11) + aux11;
00648 }
00649
00651
00652 void intpol_met_space(
00653
       met t * met,
        double p,
00654
        double lon,
00656
        double lat,
00657
        double *ps,
00658
        double *pt,
00659
        double *z.
00660
        double *t.
00661
        double *u,
00662
        double *v,
00663
        double *w,
00664
        double *pv,
        double *h2o,
00665
00666
        double *o3) {
00667
00668
        double wp, wx, wy;
00669
00670
        int ip, ix, iy;
00671
        /* Check longitude... */
if (met->lon[met->nx - 1] > 180 && lon < 0)
00672
00673
00674
          lon += 360;
00675
        /* Get indices... */
00676
00677
        ip = locate_irr(met->p, met->np, p);
        ix = locate_reg(met->lon, met->nx, lon);
00678
00679
        iy = locate_reg(met->lat, met->ny, lat);
00681
        wp = (met - p[ip + 1] - p) / (met - p[ip + 1] - met - p[ip]);
00682
        wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);
00683
00684
00685
00686
        /* Interpolate... */
00687
        if (ps != NULL)
00688
          intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
00689
        if (pt != NULL)
00690
          intpol_met_2d(met->pt, ix, iy, wx, wy, pt);
00691
        if (z != NULL)
00692
          intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy, z);
00693
        if (t != NULL)
          intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
00694
00695
        if (u != NULL)
        intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
if (v != NULL)
00696
00697
```

```
intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
00699
       if (w != NULL)
00700
         intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
       if (pv != NULL)
00701
00702
         intpol_met_3d(met->pv, ip, ix, iy, wp, wx, wy, pv);
00703
       if (h2o != NULL)
         intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
00704
00705
       if (o3 != NULL)
00706
         intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00707 }
00708
00710
00711 void intpol_met_time(
00712
       met_t * met0,
00713
       met_t * met1,
00714
       double ts,
       double p, double lon,
00715
00716
00717
       double lat,
       double *ps,
00718
       double *pt,
00719
00720
       double *z,
00721
       double *t.
00722
       double *u,
00723
       double *v,
00724
       double *w,
       double *pv,
00725
       double *h2o,
00726
00727
       double *o3) {
00728
00729
       double h2o0, h2o1, o30, o31, ps0, ps1, pt0, pt1, pv0, pv1, t0, t1, u0, u1,
00730
         v0, v1, w0, w1, wt, z0, z1;
00731
       00732
00733
00734
00735
                        pt == NULL ? NULL : &pt0,
00736
                        z == NULL ? NULL : &z0,
00737
                        t == NULL ? NULL : &t0,
00738
                        u == NULL ? NULL : &u0,
                        v == NULL ? NULL : &v0,
00739
                        w == NULL ? NULL : &w0,
00740
                        pv == NULL ? NULL : &pv0,
00741
00742
                        h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00743
       intpol_met_space(met1, p, lon, lat,
                      ps == NULL ? NULL : &ps1,
pt == NULL ? NULL : &pt1,
00744
00745
00746
                        z == NULL ? NULL : &z1,
00747
                        t == NULL ? NULL : &t1,
00748
                        u == NULL ? NULL : &u1,
00749
                        v == NULL ? NULL : &v1,
00750
                        w == NULL ? NULL : &w1,
                        pv == NULL ? NULL : &pv1,
h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00751
00752
00753
00754
       /* Get weighting factor... */
00755
       wt = (met1->time - ts) / (met1->time - met0->time);
00756
00757
       /* Interpolate... */
       if (ps != NULL)
  *ps = wt * (ps0 - ps1) + ps1;
00758
00759
00760
       if (pt != NULL)
00761
         *pt = wt * (pt0 - pt1) + pt1;
       if (z != NULL)
00762
00763
         *z = wt * (z0 - z1) + z1;
00764
       if (t != NULL)
00765
         *t = wt * (t0 - t1) + t1;
       if (u != NULL)
00766
00767
         *u = wt * (u0 - u1) + u1;
00768
       if (v != NULL)
00769
         *v = wt * (v0 - v1) + v1;
       if (w != NULL)
00770
00771
         *w = wt * (w0 - w1) + w1;
00772
       if (pv != NULL)
00773
         *pv = wt * (pv0 - pv1) + pv1;
       if (h2o != NULL)
00774
00775
       *h2o = wt * (h2o0 - h2o1) + h2o1;
if (o3 != NULL)
00776
00777
         *03 = wt * (030 - 031) + 031;
00778 }
00781
00782 void jsec2time(
00783 double jsec,
00784
      int *vear.
```

```
00785
       int *mon,
00786
       int *day,
00787
       int *hour,
00788
      int *min,
00789
       int *sec,
00790
       double *remain) {
00791
00792
       struct tm t0, *t1;
00793
00794
      time_t jsec0;
00795
00796
       t0.tm_year = 100;
00797
       t0.tm_mon = 0;
00798
       t0.tm_mday = 1;
00799
       t0.tm\_hour = 0;
      t0.tm_min = 0;
t0.tm_sec = 0;
00800
00801
00802
00803
       jsec0 = (time_t) jsec + timegm(&t0);
00804
      t1 = gmtime(&jsec0);
00805
00806
       *year = t1->tm_year + 1900;
      *mon = t1->tm_mon + 1;
*day = t1->tm_mday;
00807
00808
00809
       *hour = t1->tm_hour;
00810
       *min = t1->tm_min;
00811
       *sec = t1->tm_sec;
00812
       *remain = jsec - floor(jsec);
00813 }
00814
00816
00817 int locate_irr(
00818
      double *xx,
00819
       int n,
00820
      double x) {
00821
00822
      int i, ilo, ihi;
00823
00824
      ilo = 0;
00825
      ihi = n - 1;
      i = (ihi + ilo) >> 1;
00826
00827
00828
       if (xx[i] < xx[i + 1])
       while (ihi > ilo + 1) {
   i = (ihi + ilo) >> 1;
00829
00830
00831
          if (xx[i] > x)
00832
           ihi = i;
          else
00833
           ilo = i;
00834
00835
      } else
00836
       while (ihi > ilo + 1) {
          i = (ihi + ilo) >> 1;
00837
          if (xx[i] <= x)
00838
00839
            ihi = i;
00840
          else
00841
            ilo = i;
00842
        }
00843
00844
      return ilo;
00845 }
00846
00848
00849 int locate_reg(
00850 double *xx,
00851
       int n,
00852
      double x) {
00853
00854
      int i;
00855
      /* Calculate index... */
i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
00856
00857
00858
00859
       /* Check range... */
00860
      if (i < 0)
00861
        i = 0;
00862
      else if (i \ge n - 2)
00863
        i = n - 2;
00864
00865
       return i;
00866 }
00867
00869
00870 void read_atm(
00871
      const char *filename,
```

```
00872
        ctl_t * ctl,
atm_t * atm) {
00873
00874
00875
        FILE *in:
00876
00877
        char line[LEN], *tok;
00878
00879
        double t0;
00880
        int dimid, ip, iq, ncid, varid;
00881
00882
00883
        size t nparts:
00884
00885
        /* Init... */
00886
        atm->np = 0;
00887
00888
        /* Write info... */
00889
        printf("Read atmospheric data: sn", filename);
00890
00891
        /* Read ASCII data... */
00892
        if (ctl->atm_type == 0) {
00893
00894
           /* Open file... */
          if (!(in = fopen(filename, "r")))
00895
00896
            ERRMSG("Cannot open file!");
00897
          /* Read line... */
00898
00899
          while (fgets(line, LEN, in)) {
00900
            00901
00902
00903
00904
00905
00906
00907
00908
             /* Convert altitude to pressure... */
00910
            atm \rightarrow p[atm \rightarrow np] = P(atm \rightarrow p[atm \rightarrow np]);
00911
00912
            /* Increment data point counter... */
if ((++atm->np) > NP)
00913
              ERRMSG("Too many data points!");
00914
00915
00916
00917
           /* Close file... */
00918
          fclose(in);
00919
00920
00921
        /* Read binary data... */
        else if (ctl->atm_type == 1) {
00922
00923
00924
           /\star Open file... \star/
          if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");
00925
00926
00927
00928
           /* Read data... */
00929
          FREAD (&atm->np, int,
00930
               1,
00931
                 in);
          FREAD(atm->time, double, (size_t) atm->np,
00932
00933
00934
                 in);
00935
          FREAD(atm->p, double,
00936
                   (size_t) atm->np,
00937
                in);
          FREAD(atm->lon, double,
00938
00939
                  (size_t) atm->np,
00940
                 in);
00941
          FREAD(atm->lat, double,
00942
                   (size_t) atm->np,
00943
                in);
          for (iq = 0; iq < ctl->nq; iq++)
00944
            FREAD(atm->q[iq], double,
00945
                     (size_t) atm->np,
00946
00947
                   in);
00948
00949
           /* Close file... */
00950
          fclose(in);
00951
00952
00953
        /* Read netCDF data... */
00954
        else if (ctl->atm_type == 2) {
00955
00956
           /* Open file... */
          NC(nc_open(filename, NC_NOWRITE, &ncid));
00957
00958
```

```
/* Get dimensions... */
00960
            NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
00961
           NC(nc_inq_dimlen(ncid, dimid, &nparts));
00962
           atm->np = (int) nparts;
00963
           if (atm->np > NP)
              ERRMSG("Too many particles!");
00964
00966
00967
           NC(nc_inq_varid(ncid, "time", &varid));
           NC(nc_get_var_double(ncid, varid, &t0));
for (ip = 0; ip < atm->np; ip++)
  atm->time[ip] = t0;
00968
00969
00970
00971
           /* Read geolocations... */
NC(nc_inq_varid(ncid, "PRESS", &varid));
00972
00973
           NC(nc_get_var_double(ncid, varid, atm->p));
NC(nc_inq_varid(ncid, "LON", &varid));
00974
00975
           NC(nc_get_var_double(ncid, varid, atm->lon));
NC(nc_inq_varid(ncid, "LAT", &varid));
00976
00978
           NC(nc_get_var_double(ncid, varid, atm->lat));
00979
00980
            /* Read variables... */
00981
           if (ctl->qnt_p >= 0)
              if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
00982
00983
              NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
(ctl->qnt_t >= 0)
00985
                 (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
00986
                NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
00987
           if (ctl->qnt_u >= 0)
00988
              if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
           NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
if (ctl->qnt_v >= 0)
00989
00990
00991
              if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
                \label{local_noise} \mbox{NC(nc\_get\_var\_double(ncid, varid, atm->q[ctl->qnt\_v]));}
00992
00993
            if (ctl->qnt_w >= 0)
              if (nc_ing_varid(ncid, "W", &varid) == NC_NOERR)
00994
           NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
if (ctl->qnt_h2o >= 0)
00995
00997
              if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
00998
                NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
00999
            if (ctl->qnt_o3 >= 0)
             if (nc_inq_varid(ncid, "03", &varid) == NC_NOERR)
NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
01000
01001
           if (ctl->qnt_theta >= 0)
01002
             if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
01003
01004
                NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
01005
           if (ctl->qnt_pv >= 0)
              if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
01006
01007
01008
01009
            /* Check data... */
01010
           for (ip = 0; ip < atm->np; ip++)
01011
              if (fabs(atm->lon[ip]) > 360 \mid | fabs(atm->lat[ip]) > 90
                   || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
|| (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
|| (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
01012
01013
01014
                   || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
                atm->time[ip] = GSL_NAN;
01016
01017
                atm->p[ip] = GSL_NAN;
01018
                atm->lon[ip] = GSL_NAN;
01019
                atm->lat[ip] = GSL_NAN;
                for (iq = 0; iq < ctl->nq; iq++)
01020
                  atm->q[iq][ip] = GSL_NAN;
01021
01022
              } else {
01023
                if (ct1->qnt_h2o >= 0)
01024
                  atm->q[ctl->qnt_h2o][ip] *= 1.608;
                if (ctl->qnt_pv >= 0)
  atm->q[ctl->qnt_pv][ip] *= 1e6;
if (atm->lon[ip] > 180)
01025
01026
01027
                  atm->lon[ip] -= 360;
01029
01030
01031
            /* Close file... */
01032
           NC(nc_close(ncid));
01033
01034
01035
         /* Error... */
01036
01037
           ERRMSG("Atmospheric data type not supported!");
01038
01039
         /\star Check number of points... \star/
01040
         if (atm->np < 1)
01041
           ERRMSG("Can not read any data!");
01042 }
01043
01045
```

```
01046 void read_ctl(
       const char *filename,
01047
01048
        int argc,
01049
        char *argv[],
01050
        ctl_t * ctl) {
01051
01052
        int ip, iq;
01053
01054
        /* Write info... */
        01055
01056
                argv[0], __DATE__, __TIME__);
01057
01058
01059
        /* Initialize quantity indices... */
01060
        ctl->qnt_ens = -1;
        ctl->qnt_m = -1;
01061
        ctl \rightarrow qnt_r = -1;
01062
        ctl->qnt_rho = -1;
01063
01064
        ctl->qnt_ps = -1;
01065
        ctl->qnt_pt = -1;
01066
        ctl->qnt_z = -1;
        ctl->qnt_p = -1;
01067
        ctl->qnt_t = -1;
01068
        ct1->qnt_u = -1;
01069
01070
        ctl->qnt_v = -1;
01071
        ctl->qnt_w = -1;
01072
        ctl->qnt_h2o = -1;
01073
        ctl->qnt_o3 = -1;
01074
        ctl->qnt\_theta = -1;
        ctl->qnt_vh = -1;
01075
01076
        ct1->qnt vz = -1;
01077
        ctl->qnt_pv = -1;
01078
        ctl->qnt_tice = -1;
01079
        ctl->qnt\_tsts = -1;
        ctl->qnt_tnat = -1;
01080
        ctl->qnt\_stat = -1;
01081
01082
01083
        /* Read quantities... */
01084
        ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
01085
        if (ctl->nq > NQ)
01086
          ERRMSG("Too many quantities!");
01087
        for (iq = 0; iq < ctl->nq; iq++) {
01088
01089
          /* Read quantity name and format... */
          scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
01090
01091
01092
                    ctl->qnt_format[iq]);
01093
          /* Try to identify quantity... */
if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
  ctl->qnt_ens = iq;
01094
01095
01097
            sprintf(ctl->qnt_unit[iq], "-");
          } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
  ctl->qnt_m = iq;
01098
01099
            sprintf(ctl->qnt_unit[iq], "kg");
01100
          less if (stromp(ctl->qnt_name[iq], "r") == 0) {
ctl->qnt_r = iq;
01101
01103
            sprintf(ctl->qnt_unit[iq], "m");
01104
          } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
01105
            ctl->qnt_rho = iq;
            sprintf(ctl->qnt_unit[iq], "kg/m^3");
01106
01107
          } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
            ctl->qnt_ps = iq;
01108
            sprintf(ctl->qnt_unit[iq], "hPa");
01109
01110
           } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
01111
             ctl->qnt_pt = iq;
            sprintf(ctl->qnt_unit[iq], "hPa");
01112
          } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
ctl->qnt_z = iq;
01113
01114
01115
            sprintf(ctl->qnt_unit[iq], "km");
01116
          } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
01117
            ctl->qnt_p = iq;
            sprintf(ctl->qnt_unit[iq], "hPa");
01118
          } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
ctl->qnt_t = iq;
01119
01120
01121
            sprintf(ctl->qnt_unit[iq], "K");
01122
          } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
             ctl->qnt_u = iq;
01123
            sprintf(ctl->qnt_unit[iq], "m/s");
01124
          less if (stromp(ctl->qnt_name[iq], "v") == 0) {
ctl->qnt_v = iq;
01125
01126
            sprintf(ctl->qnt_unit[iq], "m/s");
01127
01128
          } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
            ctl->qnt_w = iq;
01129
          sprintf(ctl->qnt_unit[iq], "hPa/s");
} else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
ctl->qnt_h2o = iq;
01130
01131
01132
```

```
sprintf(ctl->qnt_unit[iq], "1");
           } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
01134
             ctl->qnt_o3 = iq;
01135
01136
             sprintf(ctl->qnt_unit[iq], "1");
           } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
  ctl->qnt_theta = iq;
  sprintf(ctl->qnt_unit[iq], "K");
01137
01138
01139
           } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
01140
            ctl->qnt_vh = iq;
01141
01142
             sprintf(ctl->qnt_unit[iq], "m/s");
           } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
01143
            ctl->qnt_vz = iq;
01144
01145
             sprintf(ctl->qnt_unit[iq], "m/s");
           } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
01146
01147
             ctl->qnt_pv = iq;
01148
             sprintf(ctl->qnt_unit[iq], "PVU");
           } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
  ctl->qnt_tice = iq;
  sprintf(ctl->qnt_unit[iq], "K");
01149
01150
01151
01152
           } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
              ctl->qnt_tsts = iq;
01153
01154
             sprintf(ctl->qnt_unit[iq], "K");
           } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
01155
             ctl->qnt_tnat = iq;
01156
             sprintf(ctl->qnt_unit[iq], "K");
01157
01158
           } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
             ctl->qnt_stat = iq;
01159
01160
             sprintf(ctl->qnt_unit[iq], "-");
01161
01162
             scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
01163
01164
01165
         /* Time steps of simulation... */
01166
        ctl->direction =
         01167
          f (ct1->direction != -1 && ct1->direction != 1)
ERRMSG("Set DIRECTION to -1 or 1!");
01168
01169
        ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
01170
01171
01172
01173
        /* Meteorological data... */
        ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
01174
        ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL); ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
01175
01176
         ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
01177
         ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
01178
         ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
01179
        ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL); ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
01180
01181
         if (ctl->met_np > EP)
01182
01183
           ERRMSG("Too many levels!");
         for (ip = 0; ip < ctl->met_np; ip++)
01184
01185
           ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
        ctl->met_tropo
01186
        = (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "0", NULL); scan_ctl(filename, argc, argv, "MET_GEOPOT", -1, "-", ctl->met_geopot); scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
01187
01188
01190
        ctl->met dt out =
01191
           scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
01192
01193
        /* Isosurface parameters... */
01194
        ctl->isosurf
        = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
01195
01196
01197
01198
         /* Diffusion parameters... */
01199
        ctl->turb_dx_trop
          = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
01200
01201
        ctl->turb_dx_strat
01202
            = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
01203
         ctl->turb_dz_trop
01204
           = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
01205
         ctl->turb_dz_strat
           = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
01206
01207
        ctl->turb mesox =
01208
           scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
01209
01210
           scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
01211
01212
         /* Mass and life time...
        ctl->molmass = scan_ctl(filename, argc, argv, "MOLMASS", -1, "1", NULL);
01213
         ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
01215
01216
           scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
01217
01218
        /* PSC analysis... */
01219
        ctl->psc h2o = scan ctl(filename, argc, argv, "PSC H2O", -1, "4e-6", NULL);
```

```
01220 ctl->psc_hno3 =
           scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
01221
01222
         /* Output of atmospheric data... */
scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
01223
01224
      atm basename);
01225
        scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
01226
         ctl->atm_dt_out
01227
           scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
01228
         ctl->atm filter
           (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
01229
01230
         ctl->atm type =
           (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
01231
01232
01233
         /* Output of CSI data... */
01234 scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
01235
        ctl->csi dt out =
         scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->
01236
01237
      csi_obsfile);
01238 ctl->csi_obsmin =
           scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
01239
01240
         ctl->csi modmin =
           scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
01241
         ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
01242
01243
01244
         ctl->csi lon0 =
01245
           scan_ctl(filename, argc, argv, "CSI_LONO", -1, "-180", NULL);
01246
         ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
01247
01248
         ctl->csi_nx =
         (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
01249
01250
01251
         ctl->csi_ny =
01252
           (int) scan ctl(filename, argc, argv, "CSI NY", -1, "180", NULL);
01253
01254
01255
         /* Output of ensemble data... */
        scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
01256
      ens_basename);
01257
         /* Output of grid data... */
01258
01259
         scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
                    ctl->grid_basename);
01260
01261
         scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
      grid_gpfile);
01262 ctl->grid_dt_out =
           scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
01263
01264
         ctl->grid_sparse =
         (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
01265
01266
01267
01268
         ctl->grid nz =
           (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
01269
01270
         ctl->grid_lon0 =
01271
           scan_ctl(filename, argc, argv, "GRID_LONO", -1, "-180", NULL);
01272
         ctl->grid lon1
01273
            scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
         ctl->grid_nx =
01274
01275
           (int) scan ctl(filename, argc, argv, "GRID NX", -1, "360", NULL);
01276
         ctl->grid lat0 =
           scan_ctl(filename, argc, argv, "GRID_LATO", -1, "-90", NULL);
01278
         ctl->grid lat1
01279
           scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
         ctl->grid_ny =
01280
            (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
01281
01282
01283
         /* Output of profile data... */
01284
         scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
01285
                    ctl->prof_basename);
01286
         scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
prof_obsfile);
01287 ctil
         ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
01288
         ctl->prof_nz =
01289
01290
           (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
01291
         ctl->prof_lon0 =
           scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
01292
01293
         ctl->prof lon1 =
           scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
01294
         ctl->prof_nx =
01295
01296
            (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
01297
         ctl->prof_lat0 =
01298
           scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
01299
         ctl->prof lat1:
01300
           scan ctl(filename, argc, argv, "PROF LAT1", -1, "90", NULL);
```

```
ctl->prof_ny =
           (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
01302
01303
01304
         /* Output of station data... */
        scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
01305
01306
                   ctl->stat basename);
        ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
01307
01308
01309
01310 }
01311
01312 /
        ******************************
01313
01314 void read_met(
01315
        ctl_t * ctl,
01316
         char *filename,
01317
        met_t * met)
01318
01319
        char cmd[2 * LEN], levname[LEN], tstr[10];
01320
01321
        static float help[EX * EY];
01322
01323
        int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
01324
01325
        size_t np, nx, ny;
01326
01327
        /* Write info... */
01328
        printf("Read meteorological data: %s\n", filename);
01329
01330
        /* Get time from filename... */
01331
         sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
01332
         year = atoi(tstr);
01333
         sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
01334
         mon = atoi(tstr);
01335
         sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
01336
         day = atoi(tstr);
         sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
01337
         hour = atoi(tstr);
01338
01339
         time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
01340
01341
         /* Open netCDF file... */
        if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
01342
01343
01344
           /* Try to stage meteo file... */
          if (ctl->met_stage[0] != '-') {
    sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
01345
01346
             year, mon, day, hour, filename);
if (system(cmd) != 0)
01347
01348
               ERRMSG("Error while staging meteo data!");
01349
01350
01351
01352
            /* Try to open again... */
01353
           NC(nc_open(filename, NC_NOWRITE, &ncid));
01354
01355
         /* Get dimensions... */
NC(nc_inq_dimid(ncid, "lon", &dimid));
01356
01357
01358
         NC(nc_inq_dimlen(ncid, dimid, &nx));
01359
         if (nx < 2 \mid \mid nx > EX)
01360
           ERRMSG("Number of longitudes out of range!");
01361
         NC(nc_inq_dimid(ncid, "lat", &dimid));
01362
01363
         NC(nc_inq_dimlen(ncid, dimid, &ny));
01364
            (ny < 2 \mid \mid ny > EY)
01365
           ERRMSG("Number of latitudes out of range!");
01366
         sprintf(levname, "lev");
01367
         NC(nc_inq_dimid(ncid, levname, &dimid));
01368
         NC(nc_inq_dimlen(ncid, dimid, &np));
if (np == 1) {
01369
01371
          sprintf(levname, "lev_2");
01372
           NC(nc_inq_dimid(ncid, levname, &dimid));
01373
          NC(nc_inq_dimlen(ncid, dimid, &np));
01374
01375
         if (np < 2 || np > EP)
01376
           ERRMSG("Number of levels out of range!");
01377
01378
         /* Store dimensions... */
        met->np = (int) np;

met->nx = (int) nx;
01379
01380
         met->ny = (int) ny;
01381
01382
        /* Get horizontal grid... */
NC(nc_inq_varid(ncid, "lon", &varid));
01383
01384
        NC(nc_get_var_double(ncid, varid, met->lon));
NC(nc_inq_varid(ncid, "lat", &varid));
NC(nc_get_var_double(ncid, varid, met->lat));
01385
01386
01387
```

```
01388
         /* Read meteorological data... */
read_met_help(ncid, "t", "T", met, met->t, 1.0);
read_met_help(ncid, "u", "U", met, met->u, 1.0);
read_met_help(ncid, "v", "V", met, met->u, 1.0);
read_met_help(ncid, "v", "V", met, met->u, 0.01f);
read_met_help(ncid, "w", "W", met, met->u, 0.01f);
01389
01390
01391
01392
01393
         read_met_help(ncid, "q", "Q", met, met->hoo, (float) (MA / 18.01528));
read_met_help(ncid, "o3", "03", met, met->o3, (float) (MA / 48.00));
01394
01395
01396
01397
         /\star Meteo data on pressure levels... \star/
01398
         if (ctl->met_np <= 0) {</pre>
01399
01400
            /* Read pressure levels from file... */
01401
           NC(nc_inq_varid(ncid, levname, &varid));
01402
           NC(nc_get_var_double(ncid, varid, met->p));
           for (ip = 0; ip < met->np; ip++)
met->p[ip] /= 100.;
01403
01404
01405
01406
            /* Extrapolate data for lower boundary... */
01407
           read_met_extrapolate(met);
01408
01409
01410
         /* Meteo data on model levels... */
01411
         else {
01412
01413
            /* Read pressure data from file... */
01414
           read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
01415
01416
           /\star Interpolate from model levels to pressure levels... \star/
01417
           read_met_ml2pl(ctl, met, met->t);
           read_met_ml2pl(ctl, met, met->u);
01418
01419
           read_met_ml2pl(ctl, met, met->v);
01420
            read_met_ml2pl(ctl, met, met->w);
01421
           read_met_ml2pl(ctl, met, met->h2o);
01422
           read_met_ml2pl(ctl, met, met->o3);
01423
01424
           /\star Set pressure levels... \star/
01425
           met->np = ctl->met_np;
01426
           for (ip = 0; ip < met->np; ip++)
01427
             met->p[ip] = ctl->met_p[ip];
01428
01429
         /* Check ordering of pressure levels... */
01430
         for (ip = 1; ip < met->np; ip++)
  if (met->p[ip - 1] < met->p[ip])
01431
01432
01433
              ERRMSG("Pressure levels must be descending!");
01434
         01435
01436
01437
           NC(nc_get_var_float(ncid, varid, help));
01438
01439
           for (iy = 0; iy < met->ny; iy++)
01440
              for (ix = 0; ix < met->nx; ix++)
         \label{eq:met-ps[ix][iy] = help[iy * met->nx + ix] / 100.;} \\ \text{else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR} \\ \\
01441
01442
                      || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
01443
           NC(nc_get_var_float(ncid, varid, help));
01444
01445
           for (iy = 0; iy < met->ny; iy++)
01446
              for (ix = 0; ix < met->nx; ix++)
01447
               met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
01448
         1 else
           for (ix = 0; ix < met->nx; ix++)
  for (iy = 0; iy < met->ny; iy++)
01449
01450
                met \rightarrow ps[ix][iy] = met \rightarrow p[0];
01451
01452
01453
         /* Create periodic boundary conditions... */
01454
         read_met_periodic(met);
01455
01456
         /* Calculate geopotential heights... */
01457
         read_met_geopot(ctl, met);
01458
01459
         /* Calculate potential vorticity... */
01460
        read_met_pv(met);
01461
01462
         /* Calculate tropopause pressure... */
01463
        read_met_tropo(ctl, met);
01464
01465
         /* Downsampling... */
01466
        read_met_sample(ctl, met);
01467
         /* Close file...
01468
01469
        NC(nc_close(ncid));
01470 }
01471
01473
01474 void read met extrapolate(
```

```
01475
       met_t * met) {
01476
01477
        int ip, ip0, ix, iy;
01478
01479    /* Loop over columns... */ 01480    #pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
       for (ix = 0; ix < met->nx; ix++)
01481
01482
          for (iy = 0; iy < met->ny; iy++) {
01483
01484
            /* Find lowest valid data point... */
            for (ip0 = met->np - 1; ip0 >= 0; ip0--)
  if (!gsl_finite(met->t[ix][iy][ip0])
01485
01486
01487
                  || !gsl_finite(met->u[ix][iy][ip0])
01488
                   || !gsl_finite(met->v[ix][iy][ip0])
01489
                   || !gsl_finite(met->w[ix][iy][ip0]))
01490
                break;
01491
01492
            /* Extrapolate... */
            for (ip = ip0; ip >= 0; ip--) {
01493
             met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
01494
01495
              met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
01496
              met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
              met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
01497
              met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
01498
01499
01500
01501
01502 }
01503
01505
01506 void read_met_geopot(
01507
       ctl_t * ctl,
01508
       met_t * met) {
01509
       static double topo_lat[EY], topo_lon[EX], topo_z[EX][EY];
01510
01511
01512
       static int init, topo_nx = -1, topo_ny;
01513
01514
       FILE *in;
01515
01516
       char line[LEN];
01517
01518
       double data[30], lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;
01519
01520
       float help[EX][EY];
01521
       int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;
01522
01523
01524
        /* Initialize geopotential heights... */
01525 #pragma omp parallel for default(shared) private(ix,iy,ip)
01526
       for (ix = 0; ix < met->nx; ix++)
01527
         for (iy = 0; iy < met->ny; iy++)
           for (ip = 0; ip < met->np; ip++)
01528
              met->z[ix][iy][ip] = GSL_NAN;
01529
01530
01531
       /* Check filename...
01532
       if (ctl->met_geopot[0] == '-')
01533
         return;
01534
01535
        /* Read surface geopotential... */
01536
       if (!init) {
01537
          init = 1;
01538
01539
          /* Write info... */
01540
          printf("Read surface geopotential: %s\n", ctl->met_geopot);
01541
01542
          /* Open file... */
01543
          if (!(in = fopen(ctl->met_geopot, "r")))
            ERRMSG("Cannot open file!");
01545
01546
          /* Read data... */
          while (fgets(line, LEN, in))
  if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rz) == 3) {
01547
01548
              if (rlon != rlon_old) {
01549
01550
               <u>if</u> ((++topo_nx) >= EX)
01551
                  ERRMSG("Too many longitudes!");
01552
                topo_ny = 0;
01553
              rlon_old = rlon:
01554
              topo_lon[topo_nx] = rlon;
01555
              topo_lat[topo_ny] = rlat;
01557
              topo_z[topo_nx][topo_ny] = rz;
01558
              if ((++topo_ny) >= EY)
01559
                ERRMSG("Too many latitudes!");
01560
          if ((++topo_nx) >= EX)
01561
```

```
ERRMSG("Too many longitudes!");
01563
01564
          /* Close file... */
01565
          fclose(in);
01566
01567
          /* Check grid spacing... */
          if (fabs(met->lon[0] - met->lon[1]) != fabs(topo_lon[0] - topo_lon[1])
01568
              || fabs(met->lat[0] - met->lat[1]) != fabs(topo_lat[0] - topo_lat[1]))
01569
01570
            printf("Warning: Grid spacing does not match!\n");
01571
01572
        /\star Apply hydrostatic equation to calculate geopotential heights... \star/
01573
01574 #pragma omp parallel for default(shared) private(ix,iy,lon,lat,tx,ty,z0,z1,ip0,ts,ip)
01575 for (ix = 0; ix < met->nx; ix++)
01576
          for (iy = 0; iy < met->ny; iy++)
01577
01578
            /* Get surface height... */
01579
            lon = met->lon[ix];
            if (lon < topo_lon[0])</pre>
01580
              lon += 360;
01581
01582
            else if (lon > topo_lon[topo_nx - 1])
01583
              lon -= 360;
            lat = met->lat[iy];
01584
            tx = locate_reg(topo_lon, topo_nx, lon);
01585
01586
            ty = locate_reg(topo_lat, topo_ny, lat);
            z0 = LIN(topo_lon[tx], topo_z[tx][ty],
topo_lon[tx + 1], topo_z[tx + 1][ty], lon);
01587
01588
            01589
01590
            z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);
01591
01592
01593
            /* Find surface pressure level... */
01594
            ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
01595
            01596
01597
01598
01599
01600
            /* Upper part of profile... */
            met->z[ix][iy][ip0 + 1]
= (float) (z0 + RI / MA / G0 * 0.5 * (ts + met->t[ix][iy][ip0 + 1])
01601
01602
                         * \log(\text{met->ps[ix][iy]} / \text{met->p[ip0 + 1])};
01603
            for (ip = ip0 + 2; ip < met->np; ip++)
01604
01605
             met->z[ix][iy][ip]
01606
               = (float) (met->z[ix][iy][ip - 1] + RI / MA / G0
                           * 0.5 * (met->t[ix][iy][ip - 1] + met->t[ix][iy][ip])
01607
01608
                            * log(met->p[ip - 1] / met->p[ip]));
01609
          }
01610
01611 /* Smooth fields... */
01612 #pragma omp parallel for default(shared) private(ip,ix,iy,n,ix2,ix3,iy2,data)
01613
       for (ip = 0; ip < met->np; ip++) {
01614
01615
          /* Median filter... */
          for (ix = 0; ix < met->nx; ix++)
01616
            for (iy = 0; iy < met->nx; iy++) {
01617
01618
01619
              for (ix2 = ix - 2; ix2 \le ix + 2; ix2++) {
01620
               ix3 = ix2;
01621
                if (ix3 < 0)
                 ix3 += met -> nx:
01622
                if (ix3 >= met->nx)
01623
01624
                  ix3 -= met->nx;
                for (iy2 = GSL_MAX(iy - 2, 0); iy2 <= GSL_MIN(iy + 2, met->ny - 1);
01625
                     iy2++)
01626
                  if (gsl_finite(met->z[ix3][iy2][ip])) {
01627
01628
                   data[n] = met \rightarrow z[ix3][iy2][ip];
01629
                    n++;
01630
01632
              if (n > 0) {
01633
                gsl_sort(data, 1, (size_t) n);
             gsl_stats_median_from_sorted_data(data, 1, (size_t) n);
} else
                help[ix][iy] = (float)
01634
01635
01636
01637
                help[ix][iy] = GSL_NAN;
01638
           }
01639
01640
          /* Copy data... */
          for (ix = 0; ix < met->nx; ix++)
  for (iy = 0; iy < met->nx; iy++)
01641
01642
01643
              met \rightarrow z[ix][iy][ip] = help[ix][iy];
01644
01645 }
01646
       ******************************
01647 /
01648
```

```
01649 void read_met_help(
      int ncid,
01650
01651
        char *varname,
01652
       char *varname2,
01653
       met_t * met,
float dest[EX][EY][EP],
01654
01655
       float scl) {
01656
01657
       static float help[EX * EY * EP];
01658
01659
       int ip, ix, iy, varid;
01660
        /* Check if variable exists... */
01661
01662
       if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
01663
01664
01665
       /* Read data... */
NC(nc_get_var_float(ncid, varid, help));
01666
01667
01668
01669
        /* Copy and check data... */
01670 #pragma omp parallel for default(shared) private(ix,iy,ip)
01671
       for (ix = 0; ix < met->nx; ix++)
         for (iy = 0; iy < met->ny; iy++)
for (ip = 0; ip < met->np; ip++) {
01672
01673
             dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
01674
01675
              if (fabsf(dest[ix][iy][ip]) < le14f)</pre>
01676
                dest[ix][iy][ip] *= scl;
01677
              else
01678
                dest[ix][iy][ip] = GSL_NAN;
01679
01680 }
01681
01683
01684 void read_met_ml2pl(
       ctl_t * ctl,
met_t * met,
01685
01686
01687
       float var[EX][EY][EP]) {
01688
01689
       double aux[EP], p[EP], pt;
01690
01691
       int ip, ip2, ix, iv;
01692
01693
        /* Loop over columns... */
01694 #pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
01695
       for (ix = 0; ix < met->nx; ix++)
01696
         for (iy = 0; iy < met->ny; iy++) {
01697
01698
            /* Copy pressure profile... */
            for (ip = 0; ip < met->np; ip++)
  p[ip] = met->pl[ix][iy][ip];
01699
01700
01701
            /* Interpolate... */
for (ip = 0; ip < ctl->met_np; ip++) {
   pt = ctl->met_p[ip];
01702
01703
01704
01705
              if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
01706
               pt = p[0];
              else if ((pt > p[met->np - 1] && p[1] > p[0])
|| (pt < p[met->np - 1] && p[1] < p[0]))
01707
01708
             01709
01710
01711
01712
01713
01714
           /* Copy data... */
for (ip = 0; ip < ctl->met_np; ip++)
01715
01716
             var[ix][iy][ip] = (float) aux[ip];
01717
01718
01719 }
01720
01722
01723 void read_met_periodic(
01724 met_t * met) {
01725
01726
       int ip, iy;
01727
01728
        /* Check longitudes... */
       if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
01729
                   + met -> lon[1] - met -> lon[0] - 360) < 0.01))
01731
01732
01733
        /\star Increase longitude counter... \star/
        if ((++met->nx) > EX)
01734
01735
         ERRMSG("Cannot create periodic boundary conditions!");
```

```
01736
        /* Set longitude... */
met->lon[met->nx - 2] + met->lon[1] - met->
01737
01738
      lon[0];
01739
01740
         /* Loop over latitudes and pressure levels... */
01741 #pragma omp parallel for default(shared) private(iy,ip)
01742
         for (iy = 0; iy < met->ny; iy++) {
           met->ps[met->nx - 1][iy] = met->ps[0][iy];
met->pt[met->nx - 1][iy] = met->pt[0][iy];
01743
01744
            for (ip = 0; ip < met->np; ip++) {
01745
             met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
01746
01747
01748
              met \rightarrow u[met \rightarrow nx - 1][iy][ip] = met \rightarrow u[0][iy][ip];
              met \rightarrow v[met \rightarrow nx - 1][iy][ip] = met \rightarrow v[0][iy][ip];
01749
              met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
met->pv[met->nx - 1][iy][ip] = met->pv[0][iy][ip];
01750
01751
              met > h2o[met -> nx - 1][iy][ip] = met -> h2o[0][iy][ip];
01752
              met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01754
01755
01756 }
01757
         **********************************
01758 /
01759
01760 void read_met_pv(
01761
         met_t * met) {
01762
01763
         double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
01764
           dtdp, dudp, dvdp, latr, vort, pows[EP];
01765
01766
         int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
01767
01768
         /* Set powers... */
01769
         for (ip = 0; ip < met->np; ip++)
01770
           pows[ip] = pow(1000. / met->p[ip], 0.286);
01771
01772
         /* Loop over grid points... */
01773 #pragma omp parallel for default(shared)
       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)
01774
         for (ix = 0; ix < met->nx; ix++) {
01775
           /* Set indices... */
ix0 = GSL_MAX(ix - 1, 0);
01776
01777
01778
           ix1 = GSL_MIN(ix + 1, met -> nx - 1);
01779
01780
            /* Loop over grid points... */
01781
           for (iy = 0; iy < met->ny; iy++) {
01782
01783
              /* Set indices... */
              iy0 = GSL_MAX(iy - 1, 0);
01784
01785
              iy1 = GSL_MIN(iy + 1, met->ny - 1);
01786
              /* Set auxiliary variables... */
latr = GSL_MIN(GSL_MAX(met->lat[iy], -89.), 89.);
01787
01788
              dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
01789
01790
              c0 = cos(met->lat[iy0] / 180. * M_PI);
c1 = cos(met->lat[iy1] / 180. * M_PI);
cr = cos(latr / 180. * M_PI);
01791
01792
01793
              vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
01794
01795
              /* Loop over grid points... */
01797
              for (ip = 0; ip < met->np; ip++) {
01798
01799
                 /* Get gradients in longitude... */
                01800
01801
01802
                 /* Get gradients in latitude... */
                dtdy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
01804
01805
01806
                /* Set indices... */
ip0 = GSL_MAX(ip - 1, 0);
ip1 = GSL_MIN(ip + 1, met->np - 1);
01807
01808
01809
01810
01811
                 /* Get gradients in pressure... */
                dp0 = 100. * (met->p[ip] - met->p[ip0]);
dp1 = 100. * (met->p[ip1] - met->p[ip]);
01812
01813
                 if (ip != ip0 && ip != ip1) {
01814
                   denom = dp0 * dp1 * (dp0 + dp1);
01815
                   dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
- dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
01816
01817
                            + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
01818
                     / denom;
01819
                   dudp = (dp0 * dp0 * met -> u[ix][iv][ip1]
01820
```

```
- dp1 * dp1 * met->u[ix][iy][ip0]
                              + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
01822
01823
                      / denom;
                   01824
01825
01826
                      / denom;
01827
                 } else {
01828
01829
                    denom = dp0 + dp1;
                    dtdp =
01830
                    (met->t[ix][iy][ip1] * pows[ip1] -
  met->t[ix][iy][ip0] * pows[ip0]) / denom;
dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
01831
01832
01833
01834
                    dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
01835
01836
                 /* Calculate PV... */
01837
01838
                 met \rightarrow pv[ix][iy][ip] = (float)
01839
                   (1e6 * G0 *
01840
                     (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
01841
01842
            }
01843 }
01844 }
01845
01846 /
01847
01848 void read_met_sample(
01849
         ctl_t * ctl,
         met_t * met) {
01850
01851
01852
         met_t *help;
01853
01854
         float w, wsum;
01855
         int ip, ip2, ix, ix2, ix3, iy, iy2;
01856
01857
01858
          /* Check parameters... */
01859
          if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1</pre>
01860
               && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
01861
             return;
01862
         /* Allocate... */
01863
01864
         ALLOC(help, met_t, 1);
01865
01866
          /* Copy data...
         help->nx = met->nx;
help->ny = met->ny;
01867
01868
         help->np = met->np;
01869
          memcpy(help->lon, met->lon, sizeof(met->lon));
01870
01871
          memcpy(help->lat, met->lat, sizeof(met->lat));
01872
          memcpy(help->p, met->p, sizeof(met->p));
01873
         /* Smoothing... */
for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
01874
01875
           for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
    for (ip = 0; ip < met->np; ip += ctl->met_dp) {
01876
01877
01878
                 help \rightarrow ps[ix][iy] = 0;
01879
                 help \rightarrow pt[ix][iy] = 0;
01880
                 help \rightarrow z[ix][iy][ip] = 0;
                 help->t[ix][iy][ip] = 0;
01881
                 help->u[ix][iy][ip] = 0;
01882
                 help \rightarrow v[ix][iy][ip] = 0;
01883
                 help \rightarrow w[ix][iy][ip] = 0;
01884
01885
                 help \rightarrow pv[ix][iy][ip] = 0;
                 help->h2o[ix][iy][ip] = 0;
help->o3[ix][iy][ip] = 0;
01886
01887
01888
                 wsum = 0;
                 for (ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1; ix2++) {
01889
                   ix3 = ix2;
01891
                   if (ix3 < 0)
01892
                      ix3 += met->nx;
                   else if (ix3 >= met->nx)
ix3 -= met->nx;
01893
01894
01895
                   for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
01896
                       if (iy2 = GSL_MAX(iy = Ct1=>met_sy + 1, 0);
    iy2 <= GSL_MIN(iy + ct1=>met_sy - 1, met=>ny - 1); iy2++)
for (ip2 = GSL_MAX(ip - ct1=>met_sp + 1, 0);
    ip2 <= GSL_MIN(ip + ct1=>met_sp - 1, met=>np - 1); ip2++) {
    w = (float) (1.0 - fabs(ix - ix2) / ct1=>met_sx)
    * (float) (1.0 - fabs(iy - iy2) / ct1=>met_sy)
01897
01898
01899
01900
01901
                            * (float) (1.0 - fabs(ip - ip2) / ctl->met_sp);
01902
01903
                         help->ps[ix][iy] += w * met->ps[ix3][iy2];
01904
                         help \rightarrow pt[ix][iy] += w * met \rightarrow pt[ix3][iy2];
                         help->z[ix][iy][ip] += w * met->z[ix3][iy2][ip2];
help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
01905
01906
01907
                         help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
```

```
help \rightarrow v[ix][iy][ip] += w * met \rightarrow v[ix3][iy2][ip2];
01909
                      help \rightarrow w[ix][iy][ip] += w * met \rightarrow w[ix3][iy2][ip2];
                      help->pv[ix][iy][ip] += w * met->pv[ix3][iy2][ip2];
01910
                      help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
01911
                      help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
01912
                      wsum += w;
01913
01914
01915
01916
               help->ps[ix][iy] /= wsum;
               help->pt[ix][iy] /= wsum;
01917
               help->t[ix][iy][ip] /= wsum;
01918
               help->z[ix][iy][ip] /= wsum;
01919
               help->u[ix][iy][ip] /= wsum;
01920
01921
               help->v[ix][iy][ip] /= wsum;
01922
               help->w[ix][iy][ip] /= wsum;
01923
               help \rightarrow pv[ix][iy][ip] /= wsum;
               help->h2o[ix][iy][ip] /= wsum;
help->o3[ix][iy][ip] /= wsum;
01924
01925
01926
01927
          }
01928
        }
01929
        /* Downsampling... */
01930
01931
        met->nx = 0;
01932
         for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
          met->lon[met->nx] = help->lon[ix];
01933
01934
           met->ny = 0;
01935
           for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
             met->lat[met->ny] = help->lat[iy];
met->ps[met->nx][met->ny] = help->ps[ix][iy];
01936
01937
01938
             met->pt[met->nx][met->ny] = help->pt[ix][iy];
             met->np = 0;
for (ip = 0; ip < help->np; ip += ctl->met_dp) {
01939
01940
01941
               met->p[met->np] = help->p[ip];
01942
               met->z[met->nx][met->ny][met->np] = help->z[ix][iy][ip];
               met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
01943
               met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
01944
               met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
01945
01946
               met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
01947
               met->pv[met->nx][met->ny][met->np] = help->pv[ix][iy][ip];
               met - > h2o[met - > nx][met - > ny][met - > np] = help - > h2o[ix][iy][ip];
01948
               \label{eq:met-loss} $$ \operatorname{met-loss}[\operatorname{met-loss}] \ [\operatorname{met-loss}] = \operatorname{help-loss}[\operatorname{ix}] \ [\operatorname{iy}] \ [\operatorname{ip}]; $$ $$
01949
01950
               met->np++:
01951
01952
             met->ny++;
01953
01954
          met->nx++;
01955
01956
01957
         /* Free... */
01958
        free(help);
01959 }
01960
01961 /*****************************
01962
01963 void read met tropo(
01964
        ctl_t * ctl,
01965
        met_t * met) {
01966
01967
        gsl_interp_accel *acc;
01968
01969
        gsl spline *spline;
01970
01971
        double p2[400], pv[400], pv2[400], t[400], t2[400], th[400], th2[400],
01972
          z[400], z2[400];
01973
01974
        int found, ix, iy, iz, iz2;
01975
        /* Allocate... */
01976
01977
        acc = gsl_interp_accel_alloc();
01978
        spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) met->np);
01979
01980
         /* Get altitude and pressure profiles... */
        for (iz = 0; iz < met->np; iz++)
  z[iz] = Z(met->p[iz]);
01981
01982
01983
         for (iz = 0; iz <= 170; iz++) {
01984
          z2[iz] = 4.5 + 0.1 * iz;
01985
          p2[iz] = P(z2[iz]);
01986
01987
01988
        /* Do not calculate tropopause... */
01989
        if (ctl->met_tropo == 0)
01990
           for (ix = 0; ix < met->nx; ix++)
01991
             for (iy = 0; iy < met->ny; iy++)
01992
               met->pt[ix][iy] = GSL_NAN;
01993
01994
        /* Use tropopause climatology... */
```

```
else if (ctl->met_tropo == 1)
          for (ix = 0; ix < met->nx; ix++)
for (iy = 0; iy < met->ny; iy++)
01996
01997
                met->pt[ix][iy] = clim_tropo(met->time, met->lat[iy]);
01998
01999
        /* Use cold point... */
else if (ctl->met_tropo == 2) {
02000
02002
02003
            /* Loop over grid points... */
           for (ix = 0; ix < met->nx; ix++)
  for (iy = 0; iy < met->ny; iy++) {
02004
02005
02006
02007
                 /* Interpolate temperature profile... */
02008
                 for (iz = 0; iz < met->np; iz++)
02009
                   t[iz] = met->t[ix][iy][iz];
                 gsl_spline_init(spline, z, t, (size_t) met->np);
for (iz = 0; iz <= 170; iz++)</pre>
02010
02011
                  t2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02012
02014
                 /* Find minimum... */
02015
                 iz = (int) gsl_stats_min_index(t2, 1, 171);
02016
                 if (iz <= 0 || iz >= 170)
02017
                  met->pt[ix][iy] = GSL_NAN;
02018
                else
02019
                   met->pt[ix][iy] = p2[iz];
02020
02021
02022
02023
         /* Use WMO definition... */
         else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
02024
02025
02026
            /* Loop over grid points... */
02027
           for (ix = 0; ix < met->nx; ix++)
02028
              for (iy = 0; iy < met->ny; iy++) {
02029
02030
                 /\star Interpolate temperature profile... \star/
                for (iz = 0; iz < met->np; iz++)
  t[iz] = met->t[ix][iy][iz];
02031
                 gsl_spline_init(spline, z, t, (size_t) met->np);
for (iz = 0; iz <= 160; iz++)</pre>
02033
02034
02035
                   t2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02036
                /* Find 1st tropopause... */
met->pt[ix][iy] = GSL_NAN;
for (iz = 0; iz <= 140; iz++) {
02037
02038
02039
02040
                   found = 1;
                   for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
  if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
    / log(p2[iz2] / p2[iz]) > 2.0) {
02041
02042
02043
02044
                        found = 0;
                       break;
02046
02047
                   if (found) {
02048
                    if (iz > 0 && iz < 140)
02049
                       met->pt[ix][iy] = p2[iz];
02050
                     break;
02051
02052
02053
02054
                 /* Find 2nd tropopause... */
                 if (ctl->met_tropo == 4) {
  met->pt[ix][iy] = GSL_NAN;
02055
02056
02057
                   for (; iz <= 140; iz++) {
02058
                    found = 1;
                      for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)</pre>
02059
                       if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
/ log(p2[iz2] / p2[iz]) < 3.0) {
02060
02061
                          found = 0;
02062
02063
                          break:
02065
                      if (found)
02066
                        break;
02067
                   for (; iz <= 140; iz++) {
02068
02069
                     found = 1;
02070
                      for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)</pre>
02071
                       if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
02072
                             / log(p2[iz2] / p2[iz]) > 2.0) {
02073
                          found = 0:
02074
                          break:
02075
                     if (found) {
02077
                       if (iz > 0 && iz < 140)
02078
                          met->pt[ix][iy] = p2[iz];
02079
                        break;
02080
02081
                   }
```

```
02082
               }
            }
02083
02084
02085
02086
        /* Use dynamical tropopause... */
02087
        else if (ctl->met tropo == 5) {
02089
           /* Loop over grid points... */
02090
          for (ix = 0; ix < met->nx; ix++)
02091
             for (iy = 0; iy < met->ny; iy++) {
02092
02093
               /\star Interpolate potential vorticity profile... \star/
               for (iz = 0; iz < met->np; iz++)
  pv[iz] = met->pv[ix][iy][iz];
02094
02095
02096
               gsl_spline_init(spline, z, pv, (size_t) met->np);
02097
               for (iz = 0; iz \leq 160; iz++)
02098
                pv2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02099
02100
               /* Interpolate potential temperature profile... */
               for (iz = 0; iz < met->np; iz++)
02101
02102
                 th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
02103
               gsl_spline_init(spline, z, th, (size_t) met->np);
               for (iz = 0; iz <= 160; iz++)</pre>
02104
02105
                 th2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02106
               /\star Find dynamical tropopause 3.5 PVU + 380 K \star/
02108
               met->pt[ix][iy] = GSL_NAN;
02109
               for (iz = 0; iz \leq 160; iz++)
02110
                 if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
                   if (iz > 0 && iz < 160)
02111
                     met->pt[ix][iy] = p2[iz];
02112
02113
                   break;
02114
02115
             }
02116
        }
02117
02118
        else
02119
          ERRMSG("Cannot calculate tropopause!");
02120
02121
         /* Free... */
02122
        gsl_spline_free(spline);
02123
        gsl_interp_accel_free(acc);
02124 }
02125
02127
02128 double scan_ctl(
02129
        const char *filename,
02130
        int argc,
02131
        char *argv[].
        const char *varname,
02132
        int arridx,
02133
02134
        const char *defvalue,
02135
        char *value) {
02136
02137
        FILE *in = NULL;
02138
02139
        char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
02140
         msg[2 * LEN], rvarname[LEN], rval[LEN];
02141
02142
        int contain = 0. i:
02143
02144
        /* Open file... */
        if (filename[strlen(filename) - 1] != '-')
if (!(in = fopen(filename, "r")))
02145
02146
            ERRMSG("Cannot open file!");
02147
02148
        /* Set full variable name... */
02149
02150
        if (arridx >= 0) {
          sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
02152
02153
         sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
02154
02155
02156
02157
02158
        /* Read data... */
02159
        if (in != NULL)
          while (fgets(line, LEN, in))
  if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
  if (strcasecmp(rvarname, fullname1) == 0 ||
    strcasecmp(rvarname, fullname2) == 0) {
02160
02161
02162
02163
02164
                 contain = 1;
02165
                 break;
02166
        for (i = 1; i < argc - 1; i++)</pre>
02167
          if (strcasecmp(argv[i], fullname1) == 0 ||
02168
```

```
strcasecmp(argv[i], fullname2) == 0) {
02170
          sprintf(rval, "%s", argv[i + 1]);
02171
          contain = 1;
02172
          break;
02173
02174
02175
       /* Close file... */
02176
       if (in != NULL)
02177
       fclose(in);
02178
02179
       /* Check for missing variables... */
02180
       if (!contain) {
        if (strlen(defvalue) > 0)
02181
02182
          sprintf(rval, "%s", defvalue);
02183
         else {
02184
          sprintf(msg, "Missing variable s!\n", fullname1);
02185
           ERRMSG (msq);
02186
02187
02188
02189
       /* Write info... */
      printf("%s = %s\n", fullname1, rval);
02190
02191
02192
      /* Return values... */
if (value != NULL)
02193
02194
        sprintf(value, "%s", rval);
02195
       return atof(rval);
02196 }
02197
02199
02200 void time2jsec(
02201 int year,
02202
       int mon,
02203
       int day,
02204
       int hour.
02205
       int min,
       int sec,
02207
       double remain,
02208
       double *jsec) {
02209
02210
       struct tm t0, t1;
02211
02212
       t0.tm_year = 100;
       t0.tm\_mon = 0;
02213
02214
       t0.tm_mday = 1;
02215
       t0.tm\_hour = 0;
       t0.tm_min = 0;
02216
       t0.tm_sec = 0;
02217
02218
       t1.tm_year = year - 1900;
02220
       t1.tm_{mon} = mon - 1;
02221
       t1.tm_mday = day;
       t1.tm_hour = hour;
02222
02223
       t1.tm_min = min;
02224
      t1.tm_sec = sec;
02226
       *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
02227 }
02228
02230
02231 void timer(
02232
     const char *name,
02233
       int id,
02234
      int mode) {
02235
       static double starttime[NTIMER], runtime[NTIMER];
02236
02237
02238
       /* Check id... */
02239
       if (id < 0 || id >= NTIMER)
02240
        ERRMSG("Too many timers!");
02241
02242
       /* Start timer... */
02243
       if (mode == 1) {
02244
        if (starttime[id] <= 0)</pre>
02245
          starttime[id] = omp_get_wtime();
02246
          ERRMSG("Timer already started!");
02247
02248
       }
02249
02250
       /* Stop timer... */
02251
       else if (mode == 2) {
02252
        if (starttime[id] > 0) {
02253
          runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
02254
          starttime[id] = -1;
02255
```

```
02256
02257
02258
       /* Print timer... */
       else if (mode == 3) {
    printf("%s = %.3f s\n", name, runtime[id]);
02259
02260
02261
         runtime[id] = 0;
02262
02263 }
02264
02266
02267 void write atm(
02268
       const char *filename,
02269
       ctl_t * ctl,
02270
       atm_t * atm,
02271
       double t) {
02272
02273
       FILE *in, *out;
02275
       char line[LEN];
02276
02277
       double r, t0, t1;
02278
       int ip, iq, year, mon, day, hour, min, sec;
02279
02280
02281
       /* Set time interval for output... */
02282
       t0 = t - 0.5 * ctl->dt_mod;
       t1 = t + 0.5 * ct1 -> dt_mod;
02283
02284
02285
       /* Write info... */
02286
       printf("Write atmospheric data: %s\n", filename);
02287
02288
       /* Write ASCII data...
02289
       if (ctl->atm_type == 0) {
02290
         /* Check if gnuplot output is requested... */
02291
         if (ctl->atm_gpfile[0] != '-') {
02292
02294
            /* Create gnuplot pipe... */
02295
           if (!(out = popen("gnuplot", "w")))
              ERRMSG("Cannot create pipe to gnuplot!");
02296
02297
           /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
02298
02299
02300
02301
            /\star Set time string... \star/
            jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02302
02303
02304
                   year, mon, day, hour, min);
02305
02306
            /* Dump gnuplot file to pipe... */
           if (!(in = fopen(ctl->atm_gpfile, "r")))
02307
02308
             ERRMSG("Cannot open file!");
           while (fgets(line, LEN, in))
  fprintf(out, "%s", line);
02309
02310
02311
           fclose(in);
02312
02313
02314
         else {
02315
            /* Create file... */
02316
            if (!(out = fopen(filename, "w")))
02317
02318
             ERRMSG("Cannot create file!");
02319
02320
02321
          /* Write header... */
         02322
02323
                  "# $2 = altitude [km] \n"
02324
                 "# $3 = longitude [deg] \n" "# $4 = latitude [deg] \n");
02325
         02326
02327
02328
02329
02330
          /* Write data... */
02331
02332
          for (ip = 0; ip < atm->np; ip++) {
02333
           /* Check time... */
if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
02334
02335
02336
             continue;
02337
           02338
02339
           atm->lon[ip], atm->lat[ip]);
for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02340
02341
02342
```

```
fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02344
02345
            fprintf(out, "\n");
02346
          }
02347
          /* Close file... */
02348
02349
          fclose(out);
02350
02351
02352
        /* Write binary data... */
02353
       else if (ctl->atm_type == 1) {
02354
02355
          /* Create file... */
02356
         if (!(out = fopen(filename, "w")))
02357
           ERRMSG("Cannot create file!");
02358
          /* Write data... */
02359
          FWRITE(&atm->np, int,
02360
02361
                 1,
02362
                 out);
02363
          FWRITE(atm->time, double,
02364
                  (size_t) atm->np,
02365
                 out);
02366
          FWRITE(atm->p, double,
02367
                  (size_t) atm->np,
02368
                 out);
02369
          FWRITE(atm->lon, double,
02370
                  (size_t) atm->np,
02371
                 out);
         FWRITE(atm->lat, double,
02372
02373
                  (size_t) atm->np,
                 out);
02375
          for (iq = 0; iq < ctl->nq; iq++)
02376
          FWRITE(atm->q[iq], double,
02377
                    (size_t) atm->np,
02378
                   out);
02379
02380
          /* Close file... */
02381
          fclose(out);
02382
02383
       /* Error... */
02384
02385
       else
02386
          ERRMSG("Atmospheric data type not supported!");
02387 }
02388
02390
02391 void write csi(
02392
       const char *filename.
       ctl_t * ctl,
atm_t * atm,
02393
02394
02395
       double t) {
02396
02397
       static FILE *in, *out;
02398
02399
       static char line[LEN];
02400
02401
       static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
02402
          rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
02403
       static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
02404
02405
02406
02407
        if (t == ctl->t_start) {
02408
02409
         /* Check quantity index for mass... */
if (ctl->qnt_m < 0)</pre>
02410
02411
            ERRMSG("Need quantity mass!");
02412
02413
          /* Open observation data file... */
          printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
02414
          if (!(in = fopen(ctl->csi_obsfile, "r")))
02415
           ERRMSG("Cannot open file!");
02416
02417
02418
          /* Create new file... */
          printf("Write CSI data: %s\n", filename);
if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");
02419
02420
02421
02422
02423
          /* Write header... */
02424
          fprintf(out,
                  "# $1 = time [s]\n"
02425
02426
                  "# $2 = number of hits (cx) n"
02427
                  "# $3 = number of misses (cy) \n"
                  "# $4 = number of false alarms (cz)\n"
02428
02429
                  "# $5 = number of observations (cx + cy) \n"
```

```
"# $6 = number of forecasts (cx + cz)\n"
                  "# $7 = bias (forecasts/observations) [%%] \n"
02431
                   "# $8 = probability of detection (POD) [%%]\n" # $9 = false alarm rate (FAR) [%%]\n"
02432
02433
                   "# $10 = critical success index (CSI) [%%]\n\n");
02434
02435
        }
02436
02437
        /* Set time interval... */
       t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
02438
02439
02440
02441
        /* Initialize grid cells... */
02442 #pragma omp parallel for default(shared) private(ix,iy,iz)
02443 for (ix = 0; ix < ctl->csi_nx; ix++)
02444
         for (iy = 0; iy < ctl->csi_ny; iy++)
02445
            for (iz = 0; iz < ctl->csi_nz; iz++)
              modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
02446
02447
02448
        /* Read observation data... */
02449
        while (fgets(line, LEN, in)) {
02450
           /* Read data... */
02451
          if (sscanf(line, "%lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02452
02453
              5)
02454
            continue;
02455
02456
          /* Check time... */
02457
          if (rt < t0)
02458
            continue;
          if (rt > t1)
02459
02460
           break:
02461
02462
          /* Calculate indices... */
          02463
02464
          iy = (int) ((rlat - ctl->csi_lat0))
02465
                       / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02466
          iz = (int) ((rz - ctl -> csi_z0))
02467
02468
                       / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02469
          /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
02470
02471
              iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02472
02473
            continue;
02474
02475
          /* Get mean observation index... */
02476
          obsmean[ix][iy][iz] += robs;
02477
          obscount[ix][iy][iz]++;
02478
02479
02480
        /* Analyze model data... */
02481 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
02482 for (ip = 0; ip < atm->np; ip++) {
02483
          /* Check time... */
02484
02485
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
02486
            continue;
02487
02488
           /* Get indices... */
02489
          ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
                       / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02490
          iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
02491
02492
                         (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
          02493
02494
02495
02496
          /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
02497
              iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02498
02499
            continue:
02500
02501
           /\star Get total mass in grid cell... \star/
02502
          modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02503
02504
02505
        /* Analyze all grid cells... */
02506 #pragma omp parallel for default(shared) private(ix,iy,iz,dlon,dlat,lat,area)
02507
        for (ix = 0; ix < ctl->csi_nx; ix++)
         for (iy = 0; iy < ctl->csi_ny; iy++)
  for (iz = 0; iz < ctl->csi_nz; iz++) {
02508
02509
02510
               /* Calculate mean observation index... */
02512
              if (obscount[ix][iy][iz] > 0)
02513
                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
02514
              /* Calculate column density... */
02515
02516
              if (modmean[ix][iy][iz] > 0) {
```

```
dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
lat = ctl->csi_lat0 + dlat * (iy + 0.5);
02518
02519
                area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180. * cos(lat * M_PI / 180.);
modmean[ix][iy][iz] /= (le6 * area);
02520
02521
02522
02524
02525
              /* Calculate CSI... */
02526
              if (obscount[ix][iy][iz] > 0) {
                if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02527
                    modmean[ix][iy][iz] >= ctl->csi_modmin)
02528
02529
                  cx++;
02530
                else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02531
                          modmean[ix][iy][iz] < ctl->csi_modmin)
02532
                  cy++;
02533
                else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
                          modmean[ix][iy][iz] >= ctl->csi_modmin)
02534
                  cz++;
02535
02536
              }
02537
02538
02539
       /* Write output... */
02540
       if (fmod(t, ctl->csi_dt_out) == 0) {
02541
          02542
02543
                  (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,

(cx + cz > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
02544
02545
02546
02547
02548
                   (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
02549
02550
          /\star Set counters to zero... \star/
02551
          cx = cy = cz = 0;
02552
02553
       /* Close file... ∗/
02555
       if (t == ctl->t_stop)
02556
         fclose(out);
02557 }
02558
02560
02561 void write_ens(
02562
       const char *filename,
02563
       ctl_t * ctl,
       atm t * atm,
02564
02565
       double t) {
02566
02567
       static FILE *out;
02568
02569
       static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
02570
        t0, t1, x[NENS][3], xm[3];
02571
02572
       static int ip, iq;
02573
02574
       static size_t i, n;
02575
       /* Init... */
02576
02577
        if (t == ctl->t_start) {
02578
          /* Check quantities... */
02580
         if (ctl->qnt_ens < 0)</pre>
02581
            ERRMSG("Missing ensemble IDs!");
02582
02583
          /* Create new file... */
          printf("Write ensemble data: %s\n", filename);
02584
          if (!(out = fopen(filename, "w")))
02585
            ERRMSG("Cannot create file!");
02587
02588
          /* Write header... */
02589
          fprintf(out,
02590
                   "# $1 = time [s] \n"
                   "# $2 = altitude [km] \n"
02591
02592
                  "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
          02593
02594
02595
          for (iq = 0; iq < ctl->nq; iq++)
  fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
02596
02597
                     ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02599
          fprintf(out, "# \$%d = number of members\n\n", 5 + 2 * ctl->nq);
02600
02601
       /* Set time interval... */
t0 = t - 0.5 * ctl->dt_mod;
02602
02603
```

```
t1 = t + 0.5 * ctl->dt_mod;
02605
         /* Init... */
02606
02607
         ens = GSL_NAN;
         n = 0;
02608
02609
02610
          /* Loop over air parcels... */
02611
         for (ip = 0; ip < atm->np; ip++) {
02612
           02613
02614
02615
             continue:
02616
02617
            /* Check ensemble id... */
02618
            if (atm->q[ctl->qnt_ens][ip] != ens) {
02619
02620
              /* Write results... */
02621
              if (n > 0) {
02622
                 /* Get mean position... */
02624
                 xm[0] = xm[1] = xm[2] = 0;
                for (i = 0; i < n; i++) {
    xm[0] += x[i][0] / (double) n;
    xm[1] += x[i][1] / (double) n;
    xm[2] += x[i][2] / (double) n;
02625
02626
02627
02628
02629
02630
                 cart2geo(xm, &dummy, &lon, &lat);
02631
                 fprintf(out, "%.2f %g %g %g", t, Z(gsl\_stats\_mean(p, 1, n)), lon,
02632
                          lat);
02633
02634
                 /* Get quantity statistics... */
for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02635
02636
02637
                   fprintf(out, ctl->qnt\_format[iq], gsl\_stats\_mean(q[iq], 1, n));\\
02638
                 for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02639
02640
02641
02642
02643
                 fprintf(out, " lu\n", n);
02644
02645
              /* Init new ensemble... */
02646
02647
              ens = atm->q[ctl->qnt_ens][ip];
              n = 0;
02648
02649
02650
02651
            /* Save data... */
02652
            p[n] = atm->p[ip];
            geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
for (iq = 0; iq < ctl->nq; iq++)
02653
02654
            q[iq][n] = atm->q[iq][ip];
if ((++n) >= NENS)
    ERRMSG("Too many data points!");
02655
02656
02657
02658
02659
02660
         /* Write results... */
02661
         if (n > 0) {
02662
02663
            /* Get mean position... */
           xm[0] = xm[1] = xm[2] = 0;
for (i = 0; i < n; i++) {
  xm[0] += x[i][0] / (double) n;
  xm[1] += x[i][1] / (double) n;
  xm[2] += x[i][2] / (double) n;</pre>
02664
02665
02666
02667
02668
02669
           cart2geo(xm, &dummy, &lon, &lat);
fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
02670
02671
02672
            /* Get quantity statistics... */
            for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02674
02675
02676
              fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02677
            for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02678
02679
02680
              fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02681
            fprintf(out, " %lu\n", n);
02682
02683
02684
02685
         /* Close file... */
02686
         if (t == ctl->t_stop)
02687
            fclose(out);
02688 }
02689
```

```
02691
02692 void write_grid(
02693
         const char *filename,
02694
         ctl_t * ctl,
         met_t * met0,
02695
         met_t * met1,
02696
         atm_t * atm,
02697
02698
         double t) {
02699
02700
         FILE *in, *out;
02701
02702
         char line[LEN];
02703
         static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
area, rho_air, press, temp, cd, vmr, t0, t1, r;
02704
02705
02706
02707
         static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
02708
         /* Check dimensions... */
02709
         if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
    ERRMSG("Grid dimensions too large!");
02710
02711
02712
02713
         /\star Check quantity index for mass... \star/
         if (ctl->qnt m < 0)
02714
02715
           ERRMSG("Need quantity mass!");
02716
02717
         /* Set time interval for output... */
         t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
02718
02719
02720
02721
          /* Set grid box size... */
02722
         dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
         dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
02723
02724
02725
02726  /* Initialize grid... */
02727  #pragma omp parallel for default(shared) private(ix,iy,iz)
02728  for (ix = 0; ix < ctl->grid_nx; ix++)
02729
           for (iy = 0; iy < ctl->grid_ny; iy++)
02730
             for (iz = 0; iz < ctl->grid_nz; iz++)
02731
                mass[ix][iy][iz] = 0;
02732
02733
         /* Average data... */
02736
            if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {</pre>
02737
02738
              /* Get index... */
              ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
02739
02740
              iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
02741
02742
               /* Check indices... */
02743
              if (ix < 0 || ix >= ctl->grid_nx ||
   iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
02744
02745
02746
                continue;
02747
02748
               /* Add mass... */
02749
               mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02750
02751
         /* Check if gnuplot output is requested... */
if (ctl->grid_gpfile[0] != '-') {
02752
02753
02754
02755
            /* Write info... */
02756
            printf("Plot grid data: %s.png\n", filename);
02757
02758
            /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
02759
02760
              ERRMSG("Cannot create pipe to gnuplot!");
02761
02762
            /* Set plot filename... */
            fprintf(out, "set out \"%s.png\"\n", filename);
02763
02764
02765
            /* Set time string... */
            jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02766
02767
02768
                     year, mon, day, hour, min);
02769
02770
            /* Dump gnuplot file to pipe... */
02771
            if (!(in = fopen(ctl->grid_gpfile, "r")))
02772
              ERRMSG("Cannot open file!");
            while (fgets(line, LEN, in))
fprintf(out, "%s", line);
02773
02774
02775
           fclose(in);
02776
02777
```

```
02778
        else {
02779
02780
          /* Write info... */
          printf("Write grid data: %s\n", filename);
02781
02782
02783
          /* Create file... */
02784
          if (!(out = fopen(filename, "w")))
02785
             ERRMSG("Cannot create file!");
02786
02787
02788
        /* Write header... */
02789
        fprintf(out,
                 "# $1 = time [s]\n"
02790
02791
                 "# $2 = altitude [km] \n"
02792
                 "# $3 = longitude [deg] \n"
                 "# $4 = latitude [deg] \n"
02793
                 "# $5 = surface area [km^2]\n"
02794
02795
                 "# $6 = layer width [km] \n"
                 "# $7 = temperature [K]\n"
02797
                 "# $8 = \text{column density } [kg/m^2] \n"
02798
                 "# $9 = volume mixing ratio [1]\n\n");
02799
        /* Write data... */
for (ix = 0; ix < ctl->grid_nx; ix++) {
   if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
02800
02801
02802
            fprintf(out, "\n");
02803
02804
              (iy = 0; iy < ctl->grid_ny; iy++) {
            if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
  fprintf(out, "\n");
for (iz = 0; iz < ctl->grid_nz; iz++)
02805
02806
02807
02808
              if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
02809
02810
                 /* Set coordinates... */
02811
                 z = ctl - > grid_z0 + dz * (iz + 0.5);
                 lon = ctl->grid_lon0 + dlon * (ix + 0.5);
lat = ctl->grid_lat0 + dlat * (iy + 0.5);
02812
02813
02814
                 /* Get pressure and temperature... */
02816
                 press = P(z);
02817
                 intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
02818
                                  NULL, &temp, NULL, NULL, NULL, NULL, NULL, NULL);
02819
                 /* Calculate surface area... */
area = dlat * dlon * SQR(RE * M_PI / 180.)
02820
02821
                  * cos(lat * M_PI / 180.);
02822
02823
                 /* Calculate column density... */
cd = mass[ix][iy][iz] / (1e6 * area);
02824
02825
02826
02827
                 /* Calculate volume mixing ratio... */
                 rho_air = 100. * press / (RA * temp);

vmr = MA / ctl->molmass * mass[ix][iy][iz]
02828
02829
02830
                   / (rho_air * 1e6 * area * 1e3 * dz);
02831
                 02832
02833
02834
02835
02836
02837
02838
        /* Close file... */
02839
02840
        fclose(out);
02841 }
02842
02844
02845 void write prof(
02846 const char *filename,
02847
        ctl_t * ctl,
02848
        met_t * met0,
02849
        met_t * met1,
        atm_t * atm,
02850
02851
        double t) {
02852
02853
        static FILE *in, *out;
02854
02855
        static char line[LEN];
02856
        static double mass[GX][GY][GZ], obsmean[GX][GY], obsmean2[GX][GY], rt, rz,
02857
02858
         rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp,
02859
          rho_air, vmr, h2o, o3;
02860
02861
        static int obscount[GX][GY], ip, ix, iy, iz, okay;
02862
02863
        /* Init... */
02864
        if (t == ctl->t_start) {
```

```
02866
            /* Check quantity index for mass... */
02867
            if (ctl->qnt_m < 0)</pre>
             ERRMSG("Need quantity mass!");
02868
02869
02870
            /* Check dimensions... */
           if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
    ERRMSG("Grid dimensions too large!");
02871
02872
02873
           /* Open observation data file... */ printf("Read profile observation data: s\n'', ctl->prof_obsfile);
02874
02875
            if (!(in = fopen(ctl->prof_obsfile, "r")))
02876
02877
             ERRMSG("Cannot open file!");
02878
02879
            /* Create new output file... */
           printf("Write profile data: s_n", filename);
if (!(out = fopen(filename, "w")))
02880
02881
              ERRMSG("Cannot create file!");
02882
02883
02884
            /* Write header... */
02885
            fprintf(out,
                     "# $1 = time [s]\n"
"# $2 = altitude [km]\n"
02886
02887
                     "# $3 = longitude [deg]\n"
"# $4 = latitude [deg]\n"
02888
02889
                     "# $5 = pressure [hPa]\n"
02891
                     "# $6 = temperature [K] \n"
02892
                     "# $7 = volume mixing ratio [1]\n"
02893
                      "# $8 = H20 volume mixing ratio [1]\n"
                     "# \$9 = 03 volume mixing ratio [1]\n"
02894
02895
                     "# $10 = observed BT index (mean) [K]\n"
02896
                     "# $11 = observed BT index (sigma) [K]\n");
02897
           /* Set grid box size... */
02898
           dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
02899
02900
02901
02902
02903
02904
         /* Set time interval... */
02905
        t0 = t - 0.5 * ctl->dt_mod;

t1 = t + 0.5 * ctl->dt_mod;
02906
02907
02908
         /* Initialize... */
02909 #pragma omp parallel for default(shared) private(ix,iy,iz)
02910
         for (ix = 0; ix < ctl->prof_nx; ix++)
02911
          for (iy = 0; iy < ctl->prof_ny; iy++) {
02912
             obsmean[ix][iy] = 0;
              obsmean2[ix][iy] = 0;
02913
              obscount[ix][iy] = 0;
for (iz = 0; iz < ctl->prof_nz; iz++)
02914
02915
02916
                mass[ix][iy][iz] = 0;
02917
02918
         /* Read observation data... */
02919
02920
         while (fgets(line, LEN, in)) {
02921
02922
            /* Read data... */
            if (sscanf(line, "%lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02923
02924
                5)
02925
             continue;
02926
02927
            /* Check time... */
02928
           if (rt < t0)
02929
              continue;
           if (rt > t1)
02930
02931
             break;
02932
02933
           /* Calculate indices... */
           ix = (int) ((rlon - ctl->prof_lon0) / dlon);
iy = (int) ((rlat - ctl->prof_lat0) / dlat);
02935
02936
           /* Check indices... */ if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
02937
02938
02939
             continue;
02940
02941
            /* Get mean observation index... */
           obsmean[ix][iy] += robs;
obsmean2[ix][iy] += SQR(robs);
02942
02943
02944
           obscount[ix][iy]++;
02945
02946
02947
         /* Analyze model data... */
02948 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
02949
         for (ip = 0; ip < atm->np; ip++) {
02950
02951
           /* Check time... */
```

```
if (atm->time[ip] < t0 || atm->time[ip] > t1)
02953
02954
           /\star Get indices... \star/
02955
           ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
02956
02957
           iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
02959
            /* Check indices... */
02960
           if (ix < 0 || ix >= ctl->prof_nx ||
    iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
02961
02962
02963
              continue:
02964
02965
            /* Get total mass in grid cell... */
02966
           mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02967
02968
02969
         /* Extract profiles... */
         for (ix = 0; ix < ctl->prof_nx; ix++)
02971
           for (iy = 0; iy < ctl->prof_ny; iy++)
02972
              if (obscount[ix][iy] > 0) {
02973
02974
                /* Check profile... */
02975
                okay = 0;
for (iz = 0; iz < ctl->prof_nz; iz++)
02976
02977
                 if (mass[ix][iy][iz] > 0) {
                     okay = 1;
02978
02979
                    break;
02980
02981
                if (!okav)
02982
                  continue;
02983
02984
                /* Write output... */
02985
                fprintf(out, "\n");
02986
                /* Loop over altitudes... */
02987
02988
                for (iz = 0; iz < ctl->prof_nz; iz++) {
02990
                  /* Set coordinates... */
                  z = ctl->prof_z0 + dz * (iz + 0.5);
lon = ctl->prof_lon0 + dlon * (ix + 0.5);
lat = ctl->prof_lat0 + dlat * (iy + 0.5);
02991
02992
02993
02994
02995
                  /\star Get pressure and temperature... \star/
02996
                  press = P(z);
02997
                   intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
02998
                                     NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
02999
                  /* Calculate surface area... */
area = dlat * dlon * SQR(M_PI * RE / 180.)
    * cos(lat * M_PI / 180.);
03000
03001
03002
03003
03004
                  /\star Calculate volume mixing ratio... \star/
                  rho_air = 100. * press / (RA * temp);
vmr = MA / ctl->molmass * mass[ix][iy][iz]
    / (rho_air * area * dz * 1e9);
03005
03006
03007
03008
03009
                  /* Write output... */
03010
                  fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g,",
                            t, z, lon, lat, press, temp, vmr, h2o, o3, obsmean[ix][iy] / obscount[ix][iy], sqrt(obsmean2[ix][iy] / obscount[ix][iy]
03011
03012
03013
03014
                                  - SQR(obsmean[ix][iy] / obscount[ix][iy])));
03015
03016
03017
        /* Close file... */
03018
03019
         if (t == ctl->t_stop)
03020
           fclose(out);
03022
03024
03025 void write station(
03026
        const char *filename,
03027
         ctl_t * ctl,
03028
03029
        double t) {
03030
03031
        static FILE *out:
03032
03033
        static double rmax2, t0, t1, x0[3], x1[3];
03034
03035
        static int ip, iq;
03036
03037
         /* Init... */
03038
         if (t == ctl->t_start) {
```

```
03040
           /* Write info... */
03041
           printf("Write station data: %s\n", filename);
03042
           /\star Create new file... \star/
03043
03044
          if (!(out = fopen(filename, "w")))
             ERRMSG("Cannot create file!");
03045
03046
03047
           /* Write header... */
           03048
03049
                    "# $2 = altitude [km] \n"
03050
                    "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03051
           for (iq = 0; iq < ctl->nq; iq++)
  fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
        ctl->qnt_name[iq], ctl->qnt_unit[iq]);
fprintf(out, "\n");
03052
03053
03054
03055
03056
03057
           /* Set geolocation and search radius... */
03058
           geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
03059
           rmax2 = SQR(ctl->stat_r);
03060
03061
        /* Set time interval for output... */
t0 = t - 0.5 * ctl->dt_mod;
03062
03063
         t1 = t + 0.5 * ctl->dt_mod;
03064
03065
         /* Loop over air parcels... */
for (ip = 0; ip < atm->np; ip++) {
03066
03067
03068
03069
           /* Check time... */
03070
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
03071
             continue;
03072
03073
           /* Check station flag... */
           if (ctl->qnt_stat >= 0)
  if (atm->q[ctl->qnt_stat][ip])
03074
03075
               continue;
03077
03078
           /\star Get Cartesian coordinates... \star/
03079
           geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
03080
03081
           /* Check horizontal distance... */
           if (DIST2(x0, x1) > rmax2)
03082
03083
            continue;
03084
03085
           /* Set station flag... */
03086
           if (ctl->qnt_stat >= 0)
03087
             atm->q[ctl->qnt_stat][ip] = 1;
03088
           /* Write data... */
fprintf(out, "%.2f %g %g %g",
03089
03090
03091
                   atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
           for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
03092
03093
03094
             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
03095
03096
           fprintf(out, "\n");
03097
03098
        /* Close file... */
03099
        if (t == ctl->t_stop)
03100
03101
           fclose(out);
03102 }
```

5.21 libtrac.h File Reference

MPTRAC library declarations.

Data Structures

• struct ctl t

Control parameters.

struct atm_t

Atmospheric 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 intpol met 2d (double array[EX][EY], int ix, int iy, double wx, double wy, double *var)

Linear interpolation of 2-D meteorological data.

- void intpol_met_3d (float array[EX][EY][EP], int ip, int ix, int iy, double wp, double wx, double wy, double *var)

 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 *pv, 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.

void 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.

void 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.

 $\bullet \ \ \text{void } \underline{\text{time2jsec}} \ (\text{int year, int mon, int day, int hour, int min, int sec, double remain, double} \ *jsec)$

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.

 $\bullet \ \ \text{void write_grid (const char *filename, ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, double t)}\\$

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;
00036
00037 radius = NORM(x);
00038 *lat = asin(x[2] / radius) * 180 / M_PI;
00039 *lon = atan2(x[1], x[0]) * 180 / M_PI;
00040 *z = radius - RE;
00041 }
```

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

Climatology of HNO3 volume mixing ratios.

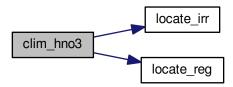
Definition at line 45 of file libtrac.c.

```
00048
00049
00050
         static double secs[12] = { 1209600.00, 3888000.00, 6393600.00,
00051
            9072000.00, 11664000.00, 14342400.00,
00052
            16934400.00, 19612800.00, 22291200.00,
00053
            24883200.00, 27561600.00, 30153600.00
00054
00055
00056
         static double lats[18] = { -85, -75, -65, -55, -45, -35, -25, -15, -5,
           5, 15, 25, 35, 45, 55, 65, 75, 85
00057
00058
00059
00060
         static double ps[10] = { 4.64159, 6.81292, 10, 14.678, 21.5443,
           31.6228, 46.4159, 68.1292, 100, 146.78
00062
00063
00064
         static double hno3[12][18][10] = {
            {(0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74}, 
{0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00065
00066
00067
              {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
              {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00068
00069
              {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
             {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37}, {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244}, {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
00070
00071
00072
             {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181}, {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00073
00075
              \{0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985\},
00076
             {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
             {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
{0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00077
00078
             {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00079
             {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00081
             {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14}
00082
              {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}}
00083
            \{\{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64\},
             {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
00084
             {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
00085
             \{0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05\},
00087
              {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661}
00088
              {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
             {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184}, {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
00089
00090
00091
             {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
             {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101}, {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00092
00093
             {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191}, {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
00094
00095
             {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11}, {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01}, {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
00096
00097
00098
             {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07}
00099
              {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63,
00100
00101
            {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00102
             {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
             {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
00103
             {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
00104
             {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
              {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33}
00106
00107
              {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174}
00108
             \{0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169\},
00109
             \{0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186\},
00110
             \{0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121\},
00111
             {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
              {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
00112
00113
              {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
             {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12}, {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82}, {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54}, {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
00114
00115
00116
              {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
00118
            {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
00119
00120
              {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00121
             \{1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5\},\
00122
             {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
00123
             {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727}
00124
             {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
```

```
{1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
                {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12}, {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
00126
00127
00128
                {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
               {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138}, {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
00129
00130
                {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02}
                {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
00132
00133
                {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
00134
                {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
                {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
00135
              {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}}, {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
00136
00137
               (0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57), {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
00138
00139
               {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37}, {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88}, {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527}, {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
00140
00141
00142
                {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
00144
                \{0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126\},
00145
00146
                {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
               {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18}, {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343}, {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
00147
00148
00149
                {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
                {0.783, 1.42, 2.65, 4.45, 6.04, 7.57,
                                                                      7.39, 5.4, 2.94, 2.25},
00151
                {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39}
00152
              {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52}, {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}}, {1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
00153
00154
00155
00156
                {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11,
                                                                        5.37, 5.12, 3.05},
                {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
00157
00158
                {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
00159
                {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
00160
                {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639}
               {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217}, {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
00161
00163
                \{0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136\},
               {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194}, {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194}, {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229}, {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302}, {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66}, {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
00164
00165
00166
00167
00168
                {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
00170
                {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9}
              {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88}, {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}}, {3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33}, {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
00171
00172
00173
00174
                {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
                {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3}, {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
00176
00177
               {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656}, {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
00178
00179
                \{1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705\},
00180
                {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12}
                {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
00182
                {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25}
00183
00184
                \{0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259\},
00185
                {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
                {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
00186
00187
                {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
                {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56}
00188
00189
                {0.735, 1.51, 2.96, 4.84, 5.92,
                                                             7.77, 7.2, 5.54, 2.56, 1.61},
              {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}, {5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4}, {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73}, {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
00190
00191
00192
00193
                {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14}
                {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38}, {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
00195
00196
00197
                {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
                {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
00198
               {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107}, {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
00199
00200
                {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
00201
00202
                {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232}
                {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
00203
                \{0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754\},
00204
                {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
00205
                {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
                \{0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5\},
00207
00208
                {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}}
00209
              \{\{1.52,\ 2.7,\ 3.79,\ 4.95,\ 3.8,\ 1.51,\ 1.11,\ 0.784,\ 1.1,\ 1.56\},
               {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74}, {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
00210
00211
```

```
{0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
               {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22}, {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
00213
00214
               {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
00215
               {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587}, {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
00216
00217
               {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
               {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
00219
00220
               {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
               {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303}, {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
00221
00222
               {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1}, {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
00223
00224
               {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
00225
00226
                {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
00227
              \{\{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91\},
                \{0.484,\ 1.38,\ 2.08,\ 3.54,\ 5.11,\ 4.98,\ 3.73,\ 2.57,\ 2.29,\ 1.84\}, 
00228
00229
               \{0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97\},
               \{0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56\},
               \{0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11\},\
00231
               {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
00232
00233
               {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
               {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968}, {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105}, {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
00234
00235
00236
               {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
               {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
00238
00239
               {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
               {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802}, 
{1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2}, 
{1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
00240
00241
00242
00243
               {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
               {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
00244
00245
              {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78}
               {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73}, {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
00246
00247
               \{0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41\},
00248
               \{0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955\},
00250
               \{0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61\},
               \{0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269\},
00251
00252
               {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
               {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121}, {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147}, {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
00253
00254
               {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172}, {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
00257
00258
               {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
               {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56}, {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
00259
00260
               {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97}, {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
00261
             {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89}}
{0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
00263
00264
               {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65}, {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28}, {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
00265
00266
00267
               {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
               \{0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256\},
00269
               {0.97, 1.75, 2.52, 3.39, 3.63, 2.48, 0.968, 0.212, 0.198}, {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173}, {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138}, {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
00270
00271
00272
00273
               {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
               \{0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6\},\
00275
00276
                {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
00277
               {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
               {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24}, {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35}, {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
00278
00279
00280
00282
00283
          double aux00, aux01, aux10, aux11, sec;
00284
00285
          int ilat, ip, isec;
00286
00287
           /* Get seconds since begin of year... */
          sec = fmod(t, 365.25 * 86400.);
00288
00289
00290
           /* Get indices... */
          isec = locate_irr(secs, 12, sec);
00291
          ilat = locate_reg(lats, 18, lat);
00292
00293
           ip = locate_irr(ps, 10, p);
00294
00295
           /* Interpolate... */
          00296
00297
00298
```

Here is the call graph for this function:



5.21.2.3 double clim_tropo (double t, double lat)

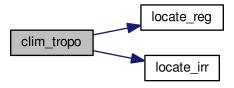
Climatology of tropopause pressure.

Definition at line 311 of file libtrac.c.

```
00313
00315
            static double doys[12]
00316
            = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00317
            static double lats[73]
00318
               = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5, -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5, -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
00319
00320
00321
              -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5, 15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5, 45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
00322
00323
00324
00325
               75, 77.5, 80, 82.5, 85, 87.5, 90
00326
00327
00328
           static double tps[12][73]
00329
             = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
00330
                       297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4, 175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
00331
                       99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54, 98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5,
00332
00333
                       152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275, 277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
00334
00335
            275.3, 275.6, 275.4, 274.1, 273.5}, {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7, 300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
00336
00337
00338
             150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
00339
00340
              98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
             98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193, 220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
00341
00342
             284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2, 287.5, 286.2, 285.8},
00343
00344
            {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
00345
00346
             297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
00347
              161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
00348
             100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
             99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8, 186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8, 279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
00349
00350
00351
00352
             304.3, 304.9, 306, 306.6, 306.2, 306},
```

```
{306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
            290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2, 195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
00354
00355
            102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
00356
            99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1, 148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
00357
00358
            263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4
             315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
00360
00361
           {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
            260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9, 205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
00362
00363
            101.8, 101.4, 101.1, 101, 101, 101.1, 101.2, 101.5, 101.9,
00364
            102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9, 165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
00365
00366
00367
            273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
00368
            325.3, 325.8, 325.8},
           (220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3, 222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2, 228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
00369
00370
             105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
00372
            106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
00373
00374
            127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
00375
            251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
           308.5, 312.2, 313.1, 313.3}, {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6, 187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
00376
00377
            235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
00379
00380
            110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
            111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4, 117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9, 224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
00381
00382
00383
00384
             275.4, 281.1,
                                286.3, 292, 296.3, 298.2, 298.8},
           {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
00385
00386
            185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
            233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7, 110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
00387
00388
            112.2, 112.5, 112.8, 113, 113, 113, 13, 114, 115.1, 116.5, 118.3, 120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
00389
00391
            230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
            278.2, 282.6, 287.4, 290.9, 292.5, 293},
00392
00393
           {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
            183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
00394
            243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5, 114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
00395
00396
            110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6, 114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
00398
00399
            203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5
           276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1}, {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5, 215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2, 237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
00400
00401
00402
            111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
00404
00405
            106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
            112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7, 206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
00406
00407
            279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
00408
             305.1},
           {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
00410
            253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
00411
00412
            223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
            108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5, 102.5, 102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
00413
00414
            109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8,
            241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6, 286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6}
00416
00417
00418
           {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
00419
            284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192, 175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
00420
            100.4, 99.96, 99.6, 99.37, 99.32, 99.31, 99.46, 99.77, 100.2, 100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160, 186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
00423
00424
            280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
00425
            281.7, 281.1, 281.2}
00426
00427
00428
           double doy, p0, p1;
00429
00430
           int imon, ilat;
00431
00432
           /* Get day of year... */
           doy = fmod(t / 86400., 365.25);
00433
           while (doy < 0)
00434
              doy += 365.25;
00435
00436
           /* Get indices... */
00437
          ilat = locate_reg(lats, 73, lat);
imon = locate_irr(doys, 12, doy);
00438
00439
```

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 451 of file libtrac.c.

```
00455

00456

00457 int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };

00458 int d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };

00459

00460 /* Get day of year... */

00461 if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))

00462 *doy = d01[mon - 1] + day - 1;

00463 else

*doy = d0[mon - 1] + day - 1;
```

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

Get date from day of year.

Definition at line 469 of file libtrac.c.

```
00473
00474
        int d0[12] = \{ 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 \};
00475
00476
        int d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00477
00478
00479
        if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
  for (i = 11; i >= 0; i--)
00480
00481
00482
           if (d01[i] <= doy)</pre>
00483
             break;
          *mon = i + 1;
00484
          *day = doy - d01[i] + 1;
00485
00486
        } else {
         for (i = 11; i >= 0; i--)
00487
          if (d0[i] <= doy)</pre>
00489
             break;
00490
          *mon = i + 1;
          *day = doy - d0[i] + 1;
00491
       }
00492
00493 }
```

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

Convert geolocation to Cartesian coordinates.

Definition at line 497 of file libtrac.c.

```
00501 {
00502
00503 double radius;
00504
00505 radius = z + RE;
00506 x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
00507 x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
00508 x[2] = radius * sin(lat / 180 * M_PI);
```

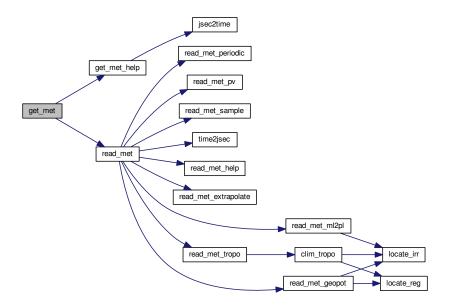
5.21.2.7 void get_met (ctl_t * ctl, char * metbase, double t, met_t ** met0, met_t ** met1)

Get meteorological data for given timestep.

Definition at line 513 of file libtrac.c.

```
00518
                         {
00520
        static int init, ip, ix, iy;
00521
        met_t *mets;
00522
00523
00524
        char filename[LEN];
00525
00526
        /* Init... */
00527
        if (t == ctl->t_start || !init) {
          init = 1;
00528
00529
          get_met_help(t, -1, metbase, ctl->dt_met, filename);
00530
00531
          read_met(ctl, filename, *met0);
00532
00533
           get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
      dt_met, filename);
00534
          read_met(ct1, filename, *met1);
00535
00536
00537
        /* Read new data for forward trajectories... */
00538
        if (t > (*met1)->time && ct1->direction == 1) {
00539
          mets = *met1;
          *met1 = *met0;
00540
          *met0 = mets;
00541
00542
          get_met_help(t, 1, metbase, ctl->dt_met, filename);
00543
          read_met(ctl, filename, *met1);
00544
00545
        /\star Read new data for backward trajectories...
00546
00547
        if (t < (*met0) -> time && ct1-> direction == -1) {
00548
         mets = *met1;
          *met1 = *met0;
00550
           *met0 = mets;
00551
           get_met_help(t, -1, metbase, ctl->dt_met, filename);
00552
          read_met(ctl, filename, *met0);
00553
00554
00555
         /* Check that grids are consistent... */
00556
        if ((*met0)->nx != (*met1)->nx
00557
             | \ (*met0) - ny != (*met1) - ny | \ (*met0) - np != (*met1) - np)
00558
          ERRMSG("Meteo grid dimensions do not match!");
        for (ix = 0; ix < (*met0)->nx; ix++)
  if ((*met0)->lon[ix] != (*met1)->lon[ix])
00559
00560
        ERRMSG("Meteo grid longitudes do not match!");
for (iy = 0; iy < (*met0)->ny; iy++)
00561
00562
00563
             ((*met0)->lat[iy] != (*met1)->lat[iy])
          if
00564
            ERRMSG("Meteo grid latitudes do not match!");
00565
        for (ip = 0; ip < (*met0)->np; ip++)
if ((*met0)->p[ip] != (*met1)->p[ip])
00566
00567
            ERRMSG("Meteo grid pressure levels do not match!");
00568 }
```

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 572 of file libtrac.c.

```
00577
00578
00579
        double t6, r;
00580
00581
        int year, mon, day, hour, min, sec;
00582
00583
        /\star Round time to fixed intervals... \star/
00584
        if (direct == -1)
          t6 = floor(t / dt_met) * dt_met;
00585
00586
        else
00587
          t6 = ceil(t / dt_met) * dt_met;
00588
00589
        /* Decode time... */
        jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00590
00591
        /* Set filename... */
sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", metbase, year, mon, day, hour);
00592
00593
00594 }
```

Here is the call graph for this function:



5.21.2.9 void intpol_met_2d (double array[EX][EY], int ix, int iy, double wx, double wy, double * var)

Linear interpolation of 2-D meteorological data.

Definition at line 598 of file libtrac.c.

```
00605
00606
         double aux00, aux01, aux10, aux11;
00607
00608
         /* Set variables...
         aux00 = array[ix][iy];
00609
         aux01 = array[ix][iy + 1];
aux10 = array[ix + 1][iy];
00611
00612
         aux11 = array[ix + 1][iy + 1];
00613
00614
         /* Interpolate horizontally... */
         aux00 = wy * (aux00 - aux01) + aux01;

aux11 = wy * (aux10 - aux11) + aux11;
00615
00616
         *var = wx * (aux00 - aux11) + aux11;
00618 }
```

5.21.2.10 void intpol_met_3d (float array[EX][EY][EP], int ip, int ix, int iy, double wp, double wx, double wy, double * var)

Linear interpolation of 3-D meteorological data.

Definition at line 622 of file libtrac.c.

```
00630
                          {
         double aux00, aux01, aux10, aux11;
00633
00634
         /* Interpolate vertically... */
         aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00635
         + array[ix][iy][ip + 1];
aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
+ array[ix][iy + 1][ip + 1];
00636
00637
00638
00639
         aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
         + array[ix + 1][iy][ip + 1];
aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
00640
00641
00642
           + array[ix + 1][iy + 1][ip + 1];
00643
         /* Interpolate horizontally... */
         aux00 = wy * (aux00 - aux01) + aux01;
aux11 = wy * (aux10 - aux11) + aux11;
00645
00646
00647
         *var = wx * (aux00 - aux11) + aux11;
00648 }
```

5.21.2.11 void intpol_met_space (met_t * met, double p, double lon, double lon, double * p, double * p

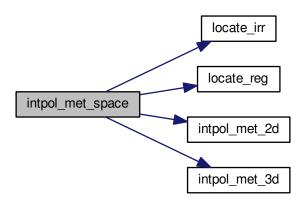
Spatial interpolation of meteorological data.

Definition at line 652 of file libtrac.c.

```
00668
        double wp, wx, wy;
00669
00670
        int ip, ix, iy;
00671
        /* Check longitude... */
if (met->lon[met->nx - 1] > 180 && lon < 0)
00672
00673
00674
          lon += 360;
00675
00676
        /* Get indices... */
00677
        ip = locate_irr(met->p, met->np, p);
00678
        ix = locate_reg(met->lon, met->nx, lon);
        iy = locate_reg(met->lat, met->ny, lat);
```

```
00681
        wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);
00682
00683
00684
00685
00686
         /* Interpolate... */
00687
         if (ps != NULL)
00688
          intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
00689
         if (pt != NULL)
           intpol_met_2d(met->pt, ix, iy, wx, wy, pt);
00690
00691
         if (z != NULL)
00692
           intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy, z);
00693
         if (t != NULL)
00694
           intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
00695
         if (u != NULL)
00696
           intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
        if (v != NULL)
00697
00698
           intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
00699
        if (w != NULL)
00700
           intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
00701
         if (pv != NULL)
00702
          intpol_met_3d(met->pv, ip, ix, iy, wp, wx, wy, pv);
00703
         if (h2o != NULL)
00704
           intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
         if (o3 != NULL)
00706
           intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00707 }
```

Here is the call graph for this function:



5.21.2.12 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)

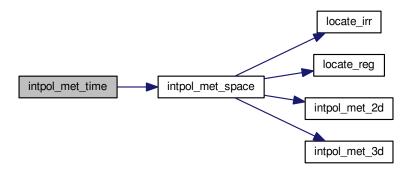
Temporal interpolation of meteorological data.

Definition at line 711 of file libtrac.c.

```
00727 {
00728
00729 double h200, h201, o30, o31, ps0, ps1, pt0, pt1, pv0, pv1, t0, t1, u0, u1,
00730 v0, v1, w0, w1, wt, z0, z1;
00731
00732 /* Spatial interpolation... */
intpol_met_space(met0, p, lon, lat,
00734 ps == NULL ? NULL : &ps0,
00735 pt == NULL ? NULL : &pt0,
00736 z == NULL ? NULL : &z0,
```

```
t == NULL ? NULL : &t0,
00738
                          u == NULL ? NULL : &u0,
00739
                          v == NULL ? NULL : &v0,
00740
                          w == NULL ? NULL : &w0,
                          pv == NULL ? NULL : &pv0,
h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00741
00742
        00743
00744
00745
                          pt == NULL ? NULL : &pt1,
00746
                          z == NULL ? NULL : &z1,
00747
                          t == NULL ? NULL : &t1,
00748
                          u == NULL ? NULL : &u1,
00749
                          v == NULL ? NULL : &v1,
00750
                          w == NULL ? NULL : &w1,
00751
                          pv == NULL ? NULL : &pv1,
00752
                          h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00753
00754
       /* Get weighting factor... */
wt = (met1->time - ts) / (met1->time - met0->time);
00755
00756
00757
        /* Interpolate... */
00758
        if (ps != NULL)
       *ps = wt * (ps0 - ps1) + ps1;
if (pt != NULL)
00759
00760
00761
          *pt = wt * (pt0 - pt1) + pt1;
00762
        if (z != NULL)
00763
          *z = wt * (z0 - z1) + z1;
        if (t != NULL)
00764
          *t = wt * (t0 - t1) + t1;
00765
        if (u != NULL)
00766
00767
          *u = wt * (u0 - u1) + u1;
00768
        if (v != NULL)
00769
          *v = wt * (v0 - v1) + v1;
        if (w != NULL)
00770
00771
          *w = wt * (w0 - w1) + w1;
        if (pv != NULL)
00772
        *pv = wt * (pv0 - pv1) + pv1;
if (h2o != NULL)
00773
00774
00775
          *h2o = wt * (h2o0 - h2o1) + h2o1;
00776
        if (o3 != NULL)
00777
          *o3 = wt * (o30 - o31) + o31;
00778 }
```

Here is the call graph for this function:



5.21.2.13 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 782 of file libtrac.c.

```
00791
00792
        struct tm t0, *t1;
00793
00794
        time_t jsec0;
00795
00796
        t0.tm_year = 100;
00797
        t0.tm_mon = 0;
        t0.tm_mday = 1;
t0.tm_hour = 0;
00798
00799
        t0.tm_min = 0;
00800
        t0.tm_sec = 0;
00801
00802
00803
        jsec0 = (time_t) jsec + timegm(&t0);
00804
        t1 = gmtime(\&jsec0);
00805
00806
        *year = t1->tm_year + 1900;
        *mon = t1->tm_mon + 1;
*day = t1->tm_mday;
00807
80800
00809
        *hour = t1->tm_hour;
00810
        *min = t1->tm_min;
        *sec = t1->tm_sec;
00811
        *remain = jsec - floor(jsec);
00812
00813 }
```

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

Find array index for irregular grid.

Definition at line 817 of file libtrac.c.

```
00820
00821
00822
        int i, ilo, ihi;
00823
00824
        ilo = 0;
        ihi = n - 1;
00825
        i = (ihi + ilo) >> 1;
00827
00828
        if (xx[i] < xx[i + 1])
         while (ihi > ilo + 1) {
  i = (ihi + ilo) >> 1;
00829
00830
             if (xx[i] > x)
00831
00832
               ihi = i;
             else
00834
               ilo = i;
00835
        } else
         while (ihi > ilo + 1) {
   i = (ihi + ilo) >> 1;
00836
00837
             <u>if</u> (xx[i] <= x)
00838
00839
               ihi = i;
00840
             else
               ilo = i;
00841
00842
          }
00843
00844
        return ilo;
00845 }
```

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

Find array index for regular grid.

Definition at line 849 of file libtrac.c.

```
00852
00853
00854
       int i:
00855
       /* Calculate index... */
00857
       i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
00858
00859
       /* Check range... */
00860
       if (i < 0)
         i = 0;
00861
00862
       else if (i >= n - 2)
00863
         i = n - 2;
00864
00865
       return i;
00866 }
```

5.21.2.16 void read_atm (const char * filename, ctl_t * ctl, atm_t * atm)

Read atmospheric data.

Definition at line 870 of file libtrac.c.

```
00873
00874
00875
          FILE *in;
00876
          char line[LEN], *tok;
00877
00878
00879
         double t0;
00880
00881
          int dimid, ip, iq, ncid, varid;
00882
00883
          size_t nparts;
00884
00885
          /* Init... */
00886
          atm->np = 0;
00887
00888
          /* Write info... */
          printf("Read atmospheric data: sn'', filename);
00889
00890
          /* Read ASCII data... */
if (ctl->atm_type == 0) {
00891
00892
00893
00894
            /\star Open file... \star/
            if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");
00895
00896
00897
00898
            /* Read line... */
00899
            while (fgets(line, LEN, in)) {
00900
               /* Read data... */
TOK(line, tok, "%lg", atm->time[atm->np]);
TOK(NULL, tok, "%lg", atm->p[atm->np]);
TOK(NULL, tok, "%lg", atm->lon[atm->np]);
TOK(NULL, tok, "%lg", atm->lat[atm->np]);
00901
00902
00903
00904
00905
                 or (iq = 0; iq < ctl->nq; iq++)
TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00906
00907
00908
00909
               /* Convert altitude to pressure... */
               atm->p[atm->np] = P(atm->p[atm->np]);
00910
00911
               /* Increment data point counter... */
if ((++atm->np) > NP)
00912
00913
00914
                 ERRMSG("Too many data points!");
00915
00916
00917
             /* Close file... */
00918
            fclose(in);
00919
00920
         /* Read binary data... */
else if (ctl->atm_type == 1) {
00921
00922
00923
00924
             /\star Open file... \star/
            if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");
00925
00926
00927
00928
             /* Read data... */
00929
            FREAD(&atm->np, int,
00930
                    1,
00931
                    in);
00932
            FREAD(atm->time, double,
00933
                      (size_t) atm->np,
                    in);
00934
00935
            FREAD(atm->p, double,
00936
                      (size_t) atm->np,
00937
                    in);
00938
            FREAD(atm->lon, double,
00939
                      (size_t) atm->np,
                    in);
00940
00941
            FREAD(atm->lat, double,
00942
                       (size_t) atm->np,
00943
                   in);
00944
            for (iq = 0; iq < ctl->nq; iq++)
00945
              FREAD(atm->q[iq], double,
00946
                        (size_t) atm->np,
00947
                      in);
00948
00949
            /* Close file... */
```

```
fclose(in);
00951
00952
00953
         /* Read netCDF data... */
00954
         else if (ctl->atm_type == 2) {
00955
            /* Open file... */
00957
           NC(nc_open(filename, NC_NOWRITE, &ncid));
00958
           /* Get dimensions... */
NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
NC(nc_inq_dimlen(ncid, dimid, &nparts));
00959
00960
00961
00962
           atm->np = (int) nparts;
           if (atm->np > NP)
00963
00964
             ERRMSG("Too many particles!");
00965
00966
           /* Get time... */
           NC(nc_inq_varid(ncid, "time", &varid));
00967
00968
           NC(nc_get_var_double(ncid, varid, &t0));
           for (ip = 0; ip < atm->np; ip++)
00970
              atm->time[ip] = t0;
00971
           /* Read geolocations... */
NC(nc_inq_varid(ncid, "PRESS", &varid));
00972
00973
00974
           NC(nc_get_var_double(ncid, varid, atm->p));
           NC(nc_inq_varid(ncid, "LON", &varid));
00975
00976
           NC(nc_get_var_double(ncid, varid, atm->lon));
           NC(nc_inq_varid(ncid, "LAT", &varid));
00977
00978
           NC(nc_get_var_double(ncid, varid, atm->lat));
00979
00980
           /* Read variables... */
00981
           if (ctl->qnt_p >= 0)
00982
                (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
00983
                \label{local_nc_delta} \mbox{NC(nc\_get\_var\_double(ncid, varid, atm->q[ctl->qnt\_p]));}
00984
              (ctl->qnt_t >= 0)
              if (nc_ing_varid(ncid, "TEMP", &varid) == NC_NOERR)
00985
           NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
if (ctl->qnt_u >= 0)
00986
00988
                (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
00989
                NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
00990
           if (ctl->qnt_v >= 0)
             if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
00991
           NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
if (ctl->qnt_w >= 0)
00992
00993
             if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
00994
00995
                NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
00996
           if (ct1->qnt_h2o >= 0)
           if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
   NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
if (ctl->qnt_o3 >= 0)
00997
00998
00999
                (nc_inq_varid(ncid, "03", &varid) == NC_NOERR)
                NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
01001
01002
           if (ctl->qnt_theta >= 0)
             if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
01003
01004
                \label{local_nc_delta} \mbox{NC(nc\_get\_var\_double(ncid, varid, atm->q[ctl->qnt\_theta]));}
           if (ctl->qnt_pv >= 0)
01005
             if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
                NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
01007
01008
           /* Check data... */
01009
           for (ip = 0; ip < atm->np; ip++)
01010
             if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
|| (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
01011
01012
                  ||(ctl->qnt_h2o) = 0 \&\& fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
01013
01014
                  | | (ctl->qnt\_theta >= 0 \&\& fabs(atm->q[ctl->qnt\_theta][ip]) > 1e10)
                  || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
01015
01016
                atm->time[ip] = GSL_NAN;
               atm->p[ip] = GSL_NAN;
atm->lon[ip] = GSL_NAN;
01017
01018
                atm->lat[ip] = GSL_NAN;
01020
                for (iq = 0; iq < ctl->nq; iq++)
01021
                  atm->q[iq][ip] = GSL_NAN;
01022
             } else {
01023
               if (ct1->qnt_h2o >= 0)
                 atm->q[ctl->qnt_h2o][ip] *= 1.608;
01024
                if (ctl->qnt_pv >= 0)
01025
                 atm->q[ctl->qnt_pv][ip] *= 1e6;
01026
01027
                if (atm->lon[ip] > 180)
01028
                  atm->lon[ip] -= 360;
01029
01030
            /* Close file... */
           NC(nc_close(ncid));
01032
01033
01034
01035
         /* Error... */
01036
        else
```

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

Read control parameters.

Definition at line 1046 of file libtrac.c.

```
01050
01051
01052
        int ip, iq;
01053
01054
        /* Write info... */
        01055
01056
                argv[0], __DATE__, __TIME__);
01057
01058
01059
        /* Initialize quantity indices... */
01060
        ctl->qnt_ens = -1;
01061
        ctl->qnt_m = -1;
        ct1->qnt_r = -1;
01062
        ctl->qnt_rho = -1;
01063
        ctl->qnt_ps = -1;
01064
        ctl->qnt_pt = -1;
01065
        ctl->qnt_z = -1;
01067
        ctl->qnt_p = -1;
        ctl->qnt_t = -1;
01068
01069
        ctl->qnt_u = -1;
        ctl->qnt_v = -1;
01070
01071
        ctl->qnt_w = -1;
01072
        ctl->qnt_h2o = -1;
01073
        ctl \rightarrow qnt_o3 = -1;
01074
        ctl->qnt\_theta = -1;
01075
        ctl->qnt_vh = -1;
        ctl->qnt_vz = -1;
01076
01077
        ctl->qnt_pv = -1;
01078
        ctl->qnt\_tice = -1;
01079
        ctl->qnt\_tsts = -1;
01080
        ctl->qnt\_tnat = -1;
01081
        ctl->qnt_stat = -1;
01082
01083
        /* Read quantities... */
        ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
01084
        if (ctl->nq > NQ)
01085
01086
          ERRMSG("Too many quantities!");
01087
        for (iq = 0; iq < ctl->nq; iq++) {
01088
          /* Read quantity name and format... */
scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
01089
01090
01091
01092
                     ctl->qnt_format[iq]);
01093
           /* Try to identify quantity... */
if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
01094
01095
            ctl->qnt_ens = iq;
01096
             sprintf(ctl->qnt_unit[iq], "-");
01098
           } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
01099
            ctl->qnt_m = iq;
            sprintf(ctl->qnt_unit[iq], "kg");
01100
          } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
ctl->qnt_r = iq;
01101
01102
             sprintf(ctl->qnt_unit[iq], "m");
01103
          } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
ctl->qnt_rho = iq;
01104
01105
             sprintf(ctl->qnt_unit[iq], "kg/m^3");
01106
          } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
  ctl->qnt_ps = iq;
  sprintf(ctl->qnt_unit[iq], "hPa");
01107
01108
01110
          } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
01111
            ctl->qnt_pt = iq;
            sprintf(ctl->qnt_unit[iq], "hPa");
01112
          } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
ctl->qnt_z = iq;
01113
01114
01115
             sprintf(ctl->qnt_unit[iq], "km");
01116
           } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
```

```
ctl->qnt_p = iq;
              sprintf(ctl->qnt_unit[iq], "hPa");
01118
            } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
01119
              ctl->qnt_t = iq;
01120
              sprintf(ctl->qnt_unit[iq], "K");
01121
            } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
  ctl->qnt_u = iq;
01122
01123
              sprintf(ctl->qnt_unit[iq], "m/s");
01124
01125
            } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
              ctl->qnt_v = iq;
01126
              sprintf(ctl->qnt_unit[iq], "m/s");
01127
            } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
01128
              ctl->qnt_w = iq;
01129
              sprintf(ctl->qnt_unit[iq], "hPa/s");
01130
           } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
01131
            ctl->qnt_h2o = iq;
01132
              sprintf(ctl->qnt_unit[iq], "1");
01133
           } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
ctl->qnt_o3 = iq;
01134
01135
01136
              sprintf(ctl->qnt_unit[iq], "1");
            } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
01137
              ctl->qnt_theta = iq;
sprintf(ctl->qnt_unit[iq], "K");
01138
01139
            } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
01140
              ctl->qnt_vh = iq;
01141
              sprintf(ctl->qnt_unit[iq], "m/s");
01142
            } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
01143
              ctl->qnt_vz = iq;
01144
01145
              sprintf(ctl->qnt_unit[iq], "m/s");
            } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
  ctl->qnt_pv = iq;
01146
01147
01148
              sprintf(ctl->qnt_unit[iq], "PVU");
            } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
01149
01150
              ctl->qnt_tice = iq;
              sprintf(ctl->qnt_unit[iq], "K");
01151
           } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
  ctl->qnt_tsts = iq;
  sprintf(ctl->qnt_unit[iq], "K");
01152
01153
01154
01155
           } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
01156
            ctl->qnt_tnat = iq;
01157
              sprintf(ctl->qnt_unit[iq], "K");
           } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
01158
             ctl->qnt_stat = iq;
01159
              sprintf(ctl->qnt_unit[iq], "-");
01160
01161
           } else
01162
              scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
01163
01164
         /* Time steps of simulation... */
01165
01166
         ctl->direction =
01167
           (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
         if (ctl->direction != -1 && ctl->direction != 1)
01168
01169
           ERRMSG("Set DIRECTION to -1 or 1!");
         ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
01170
01171
01172
         /* Meteorological data...
         ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
01174
         ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL); ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL); ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
01175
01176
01177
         ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_DF", -1, "1", NULL); ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL); ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL); ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
01178
01179
01180
01181
01182
         if (ctl->met_np > EP)
           ERRMSG("Too many levels!");
01183
         for (ip = 0; ip < ctl->met np; ip++)
01184
01185
           ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
01186
         ctl->met_tropo
         = (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "0", NULL); scan_ctl(filename, argc, argv, "MET_GEOPOT", -1, "-", ctl->met_geopot); scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
01187
01188
01189
01190
         ctl->met dt out =
01191
            scan ctl(filename, argc, argv, "MET DT OUT", -1, "0.1", NULL);
01192
01193
          /* Isosurface parameters... */
01194
         ctl->isosurf
         = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
01195
01196
01197
01198
         /* Diffusion parameters... */
01199
         ctl->turb dx trop
01200
           = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
01201
         ctl->turb_dx_strat
            = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
01202
01203
         ctl->turb_dz_trop
```

```
01204
            = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
01205
         ctl->turb dz strat
01206
           = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
01207
         ctl->turb_mesox =
           scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
01208
01209
         ctl->turb_mesoz
01210
           scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
01211
01212
          /* Mass and life time...
         ctl->molmass = scan_ctl(filename, argc, argv, "MOLMASS", -1, "1", NULL);
ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
01213
01214
01215
         ctl->tdec strat =
01216
           scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
01217
01218
         /* PSC analysis... ∗/
         ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
ctl->psc_hno3 =
01219
01220
           scan ctl(filename, argc, argv, "PSC HNO3", -1, "9e-9", NULL);
01221
01222
         /* Output of atmospheric data... */
scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
01223
01224
      atm_basename);
01225
         scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
01226
         ct.1->atm dt out =
           scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
01227
01228
         ctl->atm filter
            (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
01229
01230
         ctl->atm_type =
01231
           (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
01232
01233
         /* Output of CSI data... */
01234
         scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
01235
        ctl->csi_dt_out =
         scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->
01236
01237
       csi obsfile);
01239
           scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
01240
         ctl->csi_modmin =
01241
           scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
         ctal_ctr[filename, argc, argv, csl_mobmlw , -1, 0 , NoLL);
ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
01242
01243
01244
         ctl->csi_lon0 =
01245
01246
            scan_ctl(filename, argc, argv, "CSI_LONO", -1, "-180", NULL);
         ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
01247
01248
         ctl->csi nx =
         (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
01249
01250
01252
         ctl->csi_ny =
01253
            (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
01254
         /* Output of ensemble data... */
01255
         scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
01256
      ens basename);
01257
          /* Output of grid data... */
01258
         scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
01259
01260
                    ctl->grid basename):
         scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
01261
      grid_gpfile);
01262
       ctl->grid_dt_out =
01263
           scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
01264
         ctl->grid_sparse =
         (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
01265
01266
01267
01268
         ctl->grid_nz =
            (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
01269
01270
         ctl->grid lon0 =
01271
           scan_ctl(filename, argc, argv, "GRID_LONO", -1, "-180", NULL);
01272
         ctl->grid lon1
01273
           scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
01274
         ctl->grid nx =
01275
            (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
01276
         ctl->grid_lat0 =
           scan_ctl(filename, argc, argv, "GRID_LATO", -1, "-90", NULL);
01277
01278
         ct1->grid lat1 =
           scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
01279
         ctl->grid_ny =
01280
01281
           (int) scan ctl(filename, argc, argv, "GRID NY", -1, "180", NULL);
01282
01283
         /\star Output of profile data... \star/
         01284
01285
```

```
01286
         scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
prof_obsfile);
01287 ctl->-
         ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL); ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
01288
01289
         ctl->prof_nz =
            (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
01290
         ctl->prof_lon0
01292
            scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
01293
         ctl->prof_lon1
01294
            scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
01295
         ctl->prof_nx =
           (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
01296
01297
         ctl->prof lat0
01298
            scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
01299
         ctl->prof_lat1
01300
            scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
         ctl->prof_ny =
01301
           (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
01302
01303
01304
         /* Output of station data... */
01305
        scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
01306
                    ctl->stat_basename);
        ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
01307
01308
01309
01310 }
```

Here is the call graph for this function:



```
5.21.2.18 void read_met ( ctl_t * ctl, char * filename, met_t * met )
```

Read meteorological data file.

Definition at line 1314 of file libtrac.c.

```
01317
01318
01319
        char cmd[2 * LEN], levname[LEN], tstr[10];
01320
01321
       static float help[EX * EY];
01322
01323
       int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
01324
01325
        size_t np, nx, ny;
01326
01327
        /* Write info... */
01328
       printf("Read meteorological data: %s\n", filename);
01329
01330
        /* Get time from filename... */
01331
        sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
        year = atoi(tstr);
sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
01332
01333
01334
        mon = atoi(tstr);
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
01335
01336
        day = atoi(tstr);
01337
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
01338
        hour = atoi(tstr);
01339
        time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
01340
01341
        /* Open netCDF file... */
01342
        if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
01343
```

```
01344
             /* Try to stage meteo file... */
            if (ctl->met_stage[0] != '-') {
    sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
01345
01346
               year, mon, day, hour, filename);
if (system(cmd) != 0)
01347
01348
                 ERRMSG("Error while staging meteo data!");
01349
01350
01351
01352
             /* Try to open again... */
01353
            NC(nc_open(filename, NC_NOWRITE, &ncid));
01354
01355
          /* Get dimensions... */
NC(nc_inq_dimid(ncid, "lon", &dimid));
01356
01357
01358
          NC(nc_inq_dimlen(ncid, dimid, &nx));
01359
          if (nx < 2 \mid \mid nx > EX)
            ERRMSG("Number of longitudes out of range!");
01360
01361
          NC(nc_inq_dimid(ncid, "lat", &dimid));
01362
01363
          NC(nc_inq_dimlen(ncid, dimid, &ny));
01364
          if (ny < 2 || ny > EY)
01365
            ERRMSG("Number of latitudes out of range!");
01366
          sprintf(levname, "lev");
01367
01368
          NC(nc_inq_dimid(ncid, levname, &dimid));
          NC(nc_inq_dimlen(ncid, dimid, &np));
01369
01370
          if (np == 1) {
01371
             sprintf(levname, "lev_2");
            NC(nc_inq_dimid(ncid, levname, &dimid));
NC(nc_inq_dimlen(ncid, dimid, &np));
01372
01373
01374
01375
          if (np < 2 || np > EP)
01376
            ERRMSG("Number of levels out of range!");
01377
01378
          /* Store dimensions... */
01379
         met->np = (int) np;
met->nx = (int) nx;
01380
01381
          met->ny = (int) ny;
01382
          /* Get horizontal grid... */
NC(nc_inq_varid(ncid, "lon", &varid));
01383
01384
          NC(nc_get_var_double(ncid, varid, met->lon));
NC(nc_inq_varid(ncid, "lat", &varid));
NC(nc_get_var_double(ncid, varid, met->lat));
01385
01386
01387
01388
01389
          /* Read meteorological data... */
         /* Read meteorological data... */
read_met_help(ncid, "t", "T", met, met->t, 1.0);
read_met_help(ncid, "u", "U", met, met->u, 1.0);
read_met_help(ncid, "v", "V", met, met->v, 1.0);
read_met_help(ncid, "v", "V", met, met->v, 1.0);
read_met_help(ncid, "w", "W", met, met->v, 0.01f);
read_met_help(ncid, "q", "Q", met, met->help(ncid, "q", "Q", met, met->help(ncid, "o3", "o3", met, met->o3, (float) (MA / 48.00));
01390
01391
01392
01393
01394
01395
01396
01397
          /\star Meteo data on pressure levels... \star/
01398
          if (ctl->met_np <= 0) {</pre>
01399
01400
             /* Read pressure levels from file... */
01401
            NC(nc_inq_varid(ncid, levname, &varid));
01402
            NC(nc_get_var_double(ncid, varid, met->p));
            for (ip = 0; ip < met->np; ip++)
met->p[ip] /= 100.;
01403
01404
01405
01406
             /* Extrapolate data for lower boundary... */
01407
            read_met_extrapolate(met);
01408
01409
01410
          /* Meteo data on model levels... */
01411
          else {
01412
01413
             /* Read pressure data from file... */
01414
             read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
01415
01416
             /* Interpolate from model levels to pressure levels... */
             read_met_ml2pl(ctl, met, met->t);
01417
             read_met_ml2pl(ctl, met, met->u);
01418
             read_met_ml2pl(ctl, met, met->v);
01419
01420
             read_met_ml2pl(ctl, met, met->w);
01421
             read_met_ml2pl(ctl, met, met->h2o);
01422
             read_met_ml2pl(ctl, met, met->o3);
01423
             /* Set pressure levels... */
01424
            met->np = ctl->met_np;
for (ip = 0; ip < met->np; ip++)
01425
01426
01427
               met->p[ip] = ctl->met_p[ip];
01428
01429
01430
         /* Check ordering of pressure levels... */
```

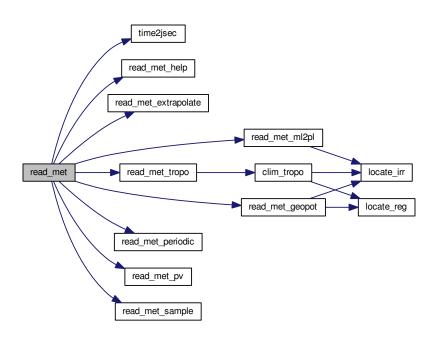
```
for (ip = 1; ip < met->np; ip++)
  if (met->p[ip - 1] < met->p[ip])
01431
01432
01433
              ERRMSG("Pressure levels must be descending!");
01434
         01435
01436
01437
01438
           NC(nc_get_var_float(ncid, varid, help));
01439
           for (iy = 0; iy < met->ny; iy++)
        for (ix = 0; ix < met->nx; ix++)

met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;

else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR

| nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
01440
01441
01442
01443
           NC(nc_get_var_float(ncid, varid, help));
for (iy = 0; iy < met->ny; iy++)
for (ix = 0; ix < met->nx; ix++)
01444
01445
01446
01447
                met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
01448
         } else
01449
           for (ix = 0; ix < met\rightarrownx; ix++)
             for (iy = 0; iy < met->ny; iy++)
01450
01451
                met \rightarrow ps[ix][iy] = met \rightarrow p[0];
01452
01453
         /* Create periodic boundary conditions... */
01454
         read_met_periodic(met);
01455
01456
         /* Calculate geopotential heights... */
01457
         read_met_geopot(ctl, met);
01458
01459
         /* Calculate potential vorticity... */
01460
         read_met_pv(met);
01461
01462
         /* Calculate tropopause pressure... */
01463
         read_met_tropo(ctl, met);
01464
01465
         /* Downsampling... */
         read_met_sample(ctl, met);
01466
01467
01468
         /* Close file... */
01469
         NC(nc_close(ncid));
01470 }
```

Here is the call graph for this function:



5.21.2.19 void read_met_extrapolate (met_t * met)

Extrapolate meteorological data at lower boundary.

Definition at line 1474 of file libtrac.c.

```
01475
01476
01477
         int ip, ip0, ix, iy;
01478
01479
         /* Loop over columns... */
01480 #pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
01481 for (ix = 0; ix < met->nx; ix++)
01482 for (iy = 0; iy < met->ny; iy++) {
01483
01484
              /* Find lowest valid data point... */
01485
              for (ip0 = met->np - 1; ip0 >= 0; ip0--)
01486
                if (!gsl_finite(met->t[ix][iy][ip0])
                    || !gsl_finite(met->u[ix][iy][ip0])
01487
01488
                    || !gsl_finite(met->v[ix][iy][ip0])
01489
                    || !gsl_finite(met->w[ix][iy][ip0]))
01490
                  break;
01491
              /* Extrapolate... */
01492
             for (ip = ip0; ip >= 0; ip--) {
01493
               met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
01494
               met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
01495
01496
               met \rightarrow v[ix][iy][ip] = met \rightarrow v[ix][iy][ip + 1];
01497
                met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
01498
               met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
01499
               met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
01500
01501
           }
01502 }
```

5.21.2.20 void read_met_geopot (ctl t * ctl, met t * met)

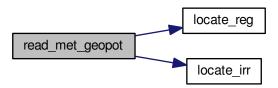
Calculate geopotential heights.

Definition at line 1506 of file libtrac.c.

```
01508
01509
01510
        static double topo lat[EY], topo lon[EX], topo z[EX][EY];
01511
01512
        static int init, topo_nx = -1, topo_ny;
01513
01514
        FILE *in:
01515
01516
        char line[LEN];
01517
01518
        double data[30], lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;
01519
01520
        float help[EX][EY];
01521
01522
        int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;
01523
01524
        /* Initialize geopotential heights... */
01525 #pragma omp parallel for default(shared) private(ix,iy,ip)
01526
        for (ix = 0; ix < met->nx; ix++)
          for (iy = 0; iy < met->ny; iy++)
  for (ip = 0; ip < met->np; ip++)
  met->z[ix][iy][ip] = GSL_NAN;
01527
01528
01529
01531
         /* Check filename...
        if (ctl->met_geopot[0] == '-')
01532
01533
          return;
01534
01535
        /* Read surface geopotential... */
01536
        if (!init) {
01537
          init = 1;
01538
01539
          /* Write info... */
01540
          printf("Read surface geopotential: %s\n", ctl->met_geopot);
01541
01542
           /* Open file... */
01543
          if (!(in = fopen(ctl->met_geopot, "r")))
01544
            ERRMSG("Cannot open file!");
01545
01546
           /* Read data... */
          while (fgets(line, LEN, in))
if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rz) == 3) {
01547
01548
01549
               if (rlon != rlon_old) {
```

```
if ((++topo_nx) >= EX)
                   ERRMSG("Too many longitudes!");
01551
01552
                topo_ny = 0;
01553
              rlon_old = rlon;
01554
01555
              topo_lon[topo_nx] = rlon;
              topo_lat[topo_ny] = rlat;
01556
01557
               topo_z[topo_nx][topo_ny] = rz;
01558
              if ((++topo_ny) >= EY)
01559
                ERRMSG("Too many latitudes!");
01560
          if ((++topo_nx) >= EX)
01561
            ERRMSG("Too many longitudes!");
01562
01563
01564
          /* Close file... */
01565
          fclose(in);
01566
01567
          /* Check grid spacing... */
01568
          if (fabs(met->lon[0] - met->lon[1]) != fabs(topo_lon[0] - topo_lon[1])
               || fabs(met->lat[0] - met->lat[1]) != fabs(topo_lat[0] - topo_lat[1]))
             printf("Warning: Grid spacing does not match!\n");
01570
01571
01572
01573
        /\star Apply hydrostatic equation to calculate geopotential heights... \star/
01574 #pragma omp parallel for default(shared) private(ix,iy,lon,lat,tx,ty,z0,z1,ip0,ts,ip)
01575 for (ix = 0; ix < met->nx; ix++)
01576
          for (iy = 0; iy < met->ny; iy++) {
01577
01578
             /* Get surface height... */
01579
            lon = met->lon[ix];
            if (lon < topo_lon[0])</pre>
01580
01581
              lon += 360;
            else if (lon > topo_lon[topo_nx - 1])
lon -= 360;
01582
01583
01584
            lat = met->lat[iy];
01585
            tx = locate_reg(topo_lon, topo_nx, lon);
            01586
01588
            z1 = LIN(topo_lon[tx], topo_z[tx][ty + 1], topo_lon[tx + 1], topo_z[tx + 1][ty + 1], lon);
01589
01590
            z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);
01591
01592
01593
            /* Find surface pressure level... */
01594
            ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
01595
01596
            /* Get surface temperature... */
            01597
01598
01599
01600
             /* Upper part of profile... */
01601
            met->z[ix][iy][ip0 + 1]
01602
              = (float) (z0 + RI / MA / G0 \star 0.5 \star (ts + met->t[ix][iy][ip0 + 1])
            * log(met->ps[ix][iy] / met->p[ip0 + 1]));
for (ip = ip0 + 2; ip < met->np; ip++)
01603
01604
              met->z[ix][iy][ip]
01605
01606
                = (float) (met->z[ix][iy][ip - 1] + RI / MA / G0
                            * 0.5 * (met->t[ix][iy][ip - 1] + met->t[ix][iy][ip])
01607
01608
                            * log(met->p[ip - 1] / met->p[ip]));
01609
          }
01610
01612  /* Smooth fields... */
01612 #pragma omp parallel for default(shared) private(ip,ix,iy,n,ix2,ix3,iy2,data)
01613  for (ip = 0; ip < met->np; ip++) {
01615
          /* Median filter... */
          for (ix = 0; ix < met->nx; ix++)
  for (iy = 0; iy < met->nx; iy++) {
01616
01617
01618
              n = 0;
              for (ix2 = ix - 2; ix2 \le ix + 2; ix2++) {
01620
                ix3 = ix2;
01621
                if (ix3 < 0)
01622
                   ix3 += met->nx;
                 if (ix3 >= met -> nx)
01623
                  ix3 -= met->nx;
01624
01625
                 for (iy2 = GSL_MAX(iy - 2, 0); iy2 <= GSL_MIN(iy + 2, met->ny - 1);
01626
                      iy2++)
01627
                   if (gsl_finite(met->z[ix3][iy2][ip])) {
01628
                    data[n] = met \rightarrow z[ix3][iy2][ip];
01629
                    n++:
                  }
01630
01631
              if (n > 0) {
01632
01633
                 gsl_sort(data, 1, (size_t) n);
01634
                 help[ix][iy] = (float)
01635
                  gsl_stats_median_from_sorted_data(data, 1, (size_t) n);
              } else
01636
```

Here is the call graph for this function:



5.21.2.21 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 1649 of file libtrac.c.

```
01655
01656
01657
         static float help[EX * EY * EP];
01658
01659
         int ip, ix, iy, varid;
01660
01661
         /* Check if variable exists... */
         if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
01662
01663
          if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
01664
01665
        /* Read data... */
NC(nc_get_var_float(ncid, varid, help));
01666
01667
01668
01669
         /* Copy and check data... */
01670 #pragma omp parallel for default(shared) private(ix,iy,ip)
01671 for (ix = 0; ix < met->nx; ix++)
01672
          for (iy = 0; iy < met->ny; iy++)
              for (ip = 0; ip < met->np; ip++) {
  dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
  if (fabsf(dest[ix][iy][ip]) < le14f)</pre>
01673
01674
01675
                  dest[ix][iy][ip] *= scl;
01677
01678
                   dest[ix][iy][ip] = GSL_NAN;
              }
01679
01680 }
```

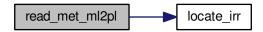
5.21.2.22 void read_met_ml2pl ($ctl_t * ctl$, $met_t * met$, float var[EX][EY][EP])

Convert meteorological data from model levels to pressure levels.

Definition at line 1684 of file libtrac.c.

```
01687
01688
01689
         double aux[EP], p[EP], pt;
01690
01691
         int ip, ip2, ix, iy;
01692
01693
         /* Loop over columns... */
01694 #pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
01695
         for (ix = 0; ix < met->nx; ix++)
01696
           for (iy = 0; iy < met->ny; iy++) {
01697
              /* Copy pressure profile... */
for (ip = 0; ip < met->np; ip++)
p[ip] = met->pl[ix][iy][ip];
01698
01699
01700
01701
              /* Interpolate... */
for (ip = 0; ip < ctl->met_np; ip++) {
   pt = ctl->met_p[ip];
   if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
01702
01703
01704
01705
01706
                   pt = p[0];
                 else if ((pt > p[met->np - 1] && p[1] > p[0])
|| (pt < p[met->np - 1] && p[1] < p[0]))
01707
01708
                01709
01710
01711
01712
01713
01714
              /* Copy data... */
for (ip = 0; ip < ctl->met_np; ip++)
01715
01716
                var[ix][iy][ip] = (float) aux[ip];
01717
01718
01719 }
```

Here is the call graph for this function:



5.21.2.23 void read_met_periodic (met_t * met)

Create meteorological data with periodic boundary conditions.

Definition at line 1723 of file libtrac.c.

```
{
01725
01726
         int ip, iy;
01727
         /* Check longitudes... */
01728
         if (!(fabs(met->lon[met->nx - 1] - met->lon[0])
01729
01730
                     + met - > lon[1] - met - > lon[0] - 360) < 0.01))
01731
01732
01733
         /\star Increase longitude counter... \star/
        if ((++met->nx) > EX)
01734
          ERRMSG("Cannot create periodic boundary conditions!");
01735
01737
        /* Set longitude... */
01738
         met \rightarrow lon[met \rightarrow nx - 1] = met \rightarrow lon[met \rightarrow nx - 2] + met \rightarrow lon[1] - met \rightarrow
      lon[0];
01739
01740
        /* Loop over latitudes and pressure levels... */
01741 #pragma omp parallel for default(shared) private(iy,ip)
        for (iy = 0; iy < met->ny; iy++) {
```

```
met - ps[met - nx - 1][iy] = met - ps[0][iy];
01744
             met->pt[met->nx - 1][iy] = met->pt[0][iy];
01745
             for (ip = 0; ip < met->np; ip++) {
               met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
01746
01747
                met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
01748
                met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
01749
01750
                met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
               met->pv[met->nx - 1][iy][ip] = met->pv[0][iy][ip];
met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01751
01752
01753
01754
01755
          }
01756 }
```

5.21.2.24 void read_met_pv (met_t * met)

Calculate potential vorticity.

Definition at line 1760 of file libtrac.c.

```
01761
01762
01763
         double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
01764
           dtdp, dudp, dvdp, latr, vort, pows[EP];
01765
01766
         int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
01767
01768
         /* Set powers... */
01769
         for (ip = 0; ip < met->np; ip++)
01770
          pows[ip] = pow(1000. / met->p[ip], 0.286);
01771
01772
         /* Loop over grid points... */
01773 #pragma omp parallel for default(shared) 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)
01774
         for (ix = 0; ix < met->nx; ix++) {
01775
           /* Set indices... */
ix0 = GSL_MAX(ix - 1, 0);
01776
01777
           ix1 = GSL_MIN(ix + 1, met \rightarrow nx - 1);
01778
01779
01780
            /* Loop over grid points... */
01781
           for (iy = 0; iy < met->ny; iy++) {
01782
             /* Set indices... */
iy0 = GSL_MAX(iy - 1, 0);
iy1 = GSL_MIN(iy + 1, met->ny - 1);
01783
01784
01785
01786
01787
              /* Set auxiliary variables... */
01788
              latr = GSL_MIN(GSL_MAX(met->lat[iy], -89.), 89.);
              01789
01790
              c0 = cos(met->lat[iy0] / 180. * M_PI);
c1 = cos(met->lat[iy1] / 180. * M_PI);
01791
01792
01793
              cr = cos(latr / 180. * M_PI);
01794
              vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
01795
01796
              /* Loop over grid points... */
for (ip = 0; ip < met->np; ip++) {
01797
01798
                 /* Get gradients in longitude... */
                dtdx = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
01800
01801
01802
                /* Get gradients in latitude... */ dtdy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
01803
01804
                dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
01805
01806
01807
                /* Set indices... */
ip0 = GSL_MAX(ip - 1, 0);
01808
                ip1 = GSL_MIN(ip + 1, met->np - 1);
01809
01810
01811
                 /* Get gradients in pressure... */
01812
                dp0 = 100. * (met -> p[ip] - met -> p[ip0]);
01813
                dp1 = 100. * (met -> p[ip1] - met -> p[ip]);
                if (ip != ip0 && ip != ip1) {
  denom = dp0 * dp1 * (dp0 + dp1);
01814
01815
                   dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
01816
01817
                            - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
01818
                            + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
```

```
/ denom;
                                                                 dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
01820
                                                                                           - dp1 * dp1 * met->u[ix][iy][ip0]
+ (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
01821
01822
                                                                        / denom;
01823
                                                                dvdp = (dp0 * dp0 * met -> v[ix][iy][ip1]
01824
                                                                                              \(\text{dp0} \times \text{dp1} \times \text{mpt} \text{mpt} \\
+ \text{(dp1} \times \text{dp1} \times \text{dp0} \times \text{dp0} \times \text{mpt} \\
+ \text{(dp1} \times \text{dp1} \times \text{dp0} \times \text{dp0} \times \text{mpt} \\
+ \text{(dp1} \times \text{dp1} \times \text{dp0} \times \text{dp0} \times \text{mpt} \\
+ \text{(dp1} \times \text{dp1} \times \text{dp0} \times \text{dp0} \times \text{mpt} \\
+ \text{(dp1} \times \text{dp1} \times \text{dp0} \times \text{dp0} \times \text{mpt} \\
+ \text{(dp1} \times \text{dp1} \times \text{dp0} \times \text{dp0} \times \text{mpt} \\
+ \text{(dp1} \times \text{dp1} \times \text{dp0} \times \text{dp0} \times \text{mpt} \\
+ \text{(dp1} \times \text{dp1} \times \text{dp0} \times \text{dp0} \times \text{mpt} \\
+ \text{(dp1} \times \text{dp1} \times \text{dp0} \times \text{dp0} \times \text{mpt} \\
+ \text{(dp1} \times \text{dp1} \times \text{dp1} \times \text{dp0} \times \text{dp1} \\
+ \text{(dp1} \times \text{dp1} \times \text{dp1} \times \text{dp1} \\
+ \text{(dp1} \times \text{dp1} \times \text{dp1} \times \text{dp1} \\
+ \text{(dp1} \times \text{dp1} \times \text{dp1} \\
+ \text{(dp1} \times \text{dp1} \times \text{dp1} \\
+ \text{(dp1} \times \text{dp1} \\
+ \text{(dp1) \times \text{dp1} \\
+ 
01825
01826
01827
                                                                       / denom;
01828
                                                       } else {
                                                                 denom = dp0 + dp1;
01829
01830
                                                                 dtdp =
                                                                    (met->t[ix][iy][ip1] * pows[ip1] -
  met->t[ix][iy][ip0] * pows[ip0]) / denom;
01831
01832
01833
                                                                 \label{eq:dudp} dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
01834
                                                                \label{eq:dvdp} dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
01835
01836
01837
                                                       /* Calculate PV... */
01838
                                                      met->pv[ix][iy][ip] = (float)
01839
                                                            (1e6 * G0 *
01840
                                                                     (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
01841
                                               }
01842
                                      }
01843
                             }
01844 }
```

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

Downsampling of meteorological data.

Definition at line 1848 of file libtrac.c.

```
01850
01851
        met_t *help;
01854
        float w, wsum;
01855
01856
        int ip, ip2, ix, ix2, ix3, iy, iy2;
01857
01858
         /* Check parameters... */
01859
        if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1</pre>
01860
              && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
01861
           return;
01862
        /* Allocate... */
ALLOC(help, met_t, 1);
01863
01864
01865
         /* Copy data... */
01866
01867
         help->nx = met->nx;
01868
        help->ny = met->ny;
         help->np = met->np;
01869
        memcpy(help->lon, met->lon, sizeof(met->lon));
memcpy(help->lat, met->lat, sizeof(met->lat));
01870
01871
01872
         memcpy(help->p, met->p, sizeof(met->p));
01873
01874
         /* Smoothing... */
         for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
01875
          for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
    for (ip = 0; ip < met->np; ip += ctl->met_dp) {
01876
                help->ps[ix][iy] = 0;
help->pt[ix][iy] = 0;
01878
01879
                help->z[ix][iy][ip] = 0;
help->t[ix][iy][ip] = 0;
01880
01881
                help->u[ix][iy][ip] = 0;
01882
                help \rightarrow v[ix][iy][ip] = 0;
01883
                help->w[ix][iy][ip] = 0;
01884
01885
                help \rightarrow pv[ix][iy][ip] = 0;
                help->h2o[ix][iy][ip] = 0;
01886
                help \rightarrow 03[ix][iy][ip] = 0;
01887
                for (ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1; ix2++) {
01888
01889
01890
                 ix3 = ix2;
01891
                  if (ix3 < 0)
01892
                     ix3 += met->nx;
                  else if (ix3 \ge met - > nx)
01893
01894
                    ix3 -= met->nx;
01895
01896
                   for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
```

```
iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
                    for (ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
01898
01899
                      w = (float) (1.0 - fabs(ix - ix2) / ctl->met_sx)
* (float) (1.0 - fabs(iy - iy2) / ctl->met_sy)
01900
01901
                         * (float) (1.0 - fabs(ip - ip2) / ctl->met_sp);
01902
                      help->ps[ix][iy] += w * met->ps[ix3][iy2];
01903
01904
                       help \rightarrow pt[ix][iy] += w * met \rightarrow pt[ix3][iy2];
01905
                      \label{eq:help-z} $$ $ help->z[ix][iy][ip] += w * met->z[ix3][iy2][ip2]; $$ $$
01906
                      help \rightarrow t[ix][iy][ip] += w * met \rightarrow t[ix3][iy2][ip2];
                      help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
01907
                      help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
01908
                      help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
01909
01910
                      help->pv[ix][iy][ip] += w * met->pv[ix3][iy2][ip2];
01911
                      help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
01912
                      \label{eq:help-o3} \verb|[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
01913
                      wsum += w;
                    }
01914
01915
01916
                help->ps[ix][iy] /= wsum;
01917
                help->pt[ix][iy] /= wsum;
01918
                help->t[ix][iy][ip] /= wsum;
                help->z[ix][iy][ip] /= wsum;
01919
                help->u[ix][iy][ip] /= wsum;
01920
                help->v[ix][iy][ip] /= wsum;
help->w[ix][iy][ip] /= wsum;
01921
01922
01923
                help->pv[ix][iy][ip] /= wsum;
01924
                help->h2o[ix][iy][ip] /= wsum;
01925
                help->o3[ix][iy][ip] /= wsum;
01926
01927
           }
01928
        }
01929
01930
         /* Downsampling... */
         met->nx = 0;
for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
01931
01932
01933
          met->lon[met->nx] = help->lon[ix];
01934
           met->ny = 0;
01935
           for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
01936
             met->lat[met->ny] = help->lat[iy];
01937
             met->ps[met->nx][met->ny] = help->ps[ix][iy];
             met->pt[met->nx][met->ny] = help->pt[ix][iy];
01938
01939
             met.->np = 0:
01940
             for (ip = 0; ip < help->np; ip += ctl->met_dp) {
               met->p[met->np] = help->p[ip];
01941
01942
                met \rightarrow z[met \rightarrow nx][met \rightarrow ny][met \rightarrow np] = help \rightarrow z[ix][iy][ip];
01943
                met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
               met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
01944
               met->v[met->nx] [met->ny] [met->np] = help->v[ix][iy][ip];
01945
               met->w[met->nx] [met->ny] [met->np] = help->w[ix][iy][ip];
01946
                met->pv[met->nx][met->ny][met->np] = help->pv[ix][iy][ip];
01947
01948
                met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
01949
                met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
01950
               met->np++;
01951
01952
             met->nv++;
01953
01954
           met->nx++;
01955
01956
01957
         /* Free... */
01958
        free(help);
01959 }
```

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

Calculate tropopause pressure.

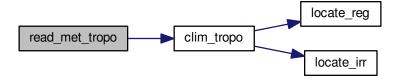
Definition at line 1963 of file libtrac.c.

```
01965
                     {
01966
01967
       gsl_interp_accel *acc;
01968
01969
       gsl_spline *spline;
01970
01971
        double p2[400], pv[400], pv2[400], t[400], t2[400], th[400], th2[400],
01972
          z[400], z2[400];
01973
01974
       int found, ix, iy, iz, iz2;
```

```
01975
01976
         /* Allocate... */
01977
         acc = gsl_interp_accel_alloc();
01978
         spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) met->np);
01979
01980
          /\star Get altitude and pressure profiles... \star/
         for (iz = 0; iz < met->np; iz++)
01982
            z[iz] = Z(met->p[iz]);
         for (iz = 0; iz <= 170; iz++) {
  z2[iz] = 4.5 + 0.1 * iz;
  p2[iz] = P(z2[iz]);</pre>
01983
01984
01985
01986
01987
01988
          /* Do not calculate tropopause... */
01989
          if (ctl->met_tropo == 0)
           for (ix = 0; ix < met->nx; ix++)
  for (iy = 0; iy < met->ny; iy++)
  met->pt[ix][iy] = GSL_NAN;
01990
01991
01992
01993
01994
          /* Use tropopause climatology... */
01995
         else if (ctl->met_tropo == 1)
01996
           for (ix = 0; ix < met->nx; ix++)
              for (iy = 0; iy < met->ny; iy++)
01997
01998
                met->pt[ix][iy] = clim_tropo(met->time, met->lat[iy]);
01999
02000
         /* Use cold point... */
02001
         else if (ctl->met_tropo == 2) {
02002
02003
            /* Loop over grid points... */
02004
            for (ix = 0; ix < met->nx; ix++)
02005
               for (iy = 0; iy < met->ny; iy++) {
02006
02007
                  /* Interpolate temperature profile... */
                 for (iz = 0; iz < met->np; iz++)
  t[iz] = met->t[ix][iy][iz];
02008
02009
                 gsl_spline_init(spline, z, t, (size_t) met->np);
for (iz = 0; iz <= 170; iz++)</pre>
02010
02011
                   t2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02013
02014
                  /* Find minimum... */
                 iz = (int) gsl_stats_min_index(t2, 1, 171);
if (iz <= 0 || iz >= 170)
  met->pt[ix][iy] = GSL_NAN;
02015
02016
02017
02018
                 else
02019
                   met->pt[ix][iy] = p2[iz];
02020
              }
02021
02022
         /* Use WMO definition... */
02023
         else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
02024
            /* Loop over grid points... */
for (ix = 0; ix < met->nx; ix++)
02026
02027
02028
              for (iy = 0; iy < met->ny; iy++) {
02029
02030
                  /* Interpolate temperature profile... */
                 for (iz = 0; iz < met->np; iz++)
02032
                   t[iz] = met \rightarrow t[ix][iy][iz];
                 gsl_spline_init(spline, z, t, (size_t) met->np);
for (iz = 0; iz <= 160; iz++)
   t2[iz] = gsl_spline_eval(spline, z2[iz], acc);</pre>
02033
02034
02035
02036
02037
                  /* Find 1st tropopause... */
02038
                 met->pt[ix][iy] = GSL_NAN;
02039
                 for (iz = 0; iz <= 140; iz++) {
                    found = 1;
02040
02041
                    for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
if (1000. * G0 / RA * log(t2[iz2] / t2[iz])</pre>
02042
                           / log(p2[iz2] / p2[iz]) > 2.0) {
02043
                        found = 0;
02045
02046
02047
                    if (found) {
                     if (iz > 0 && iz < 140)
02048
                        met->pt[ix][iy] = p2[iz];
02049
02050
                      break;
02051
                    }
02052
02053
                 /* Find 2nd tropopause... */
02054
                 if (ctl->met_tropo == 4) {
  met->pt[ix][iy] = GSL_NAN;
02055
02057
                    for (; iz <= 140; iz++) {
02058
                      found = 1;
                      for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
  if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
     / log(p2[iz2] / p2[iz]) < 3.0) {</pre>
02059
02060
02061
```

```
found = 0;
02063
                          break;
02064
02065
                     if (found)
02066
                       break;
02067
                   for (; iz <= 140; iz++) {</pre>
02069
                      found = 1;
                      for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
  if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
     / log(p2[iz2] / p2[iz]) > 2.0) {
02070
02071
02072
02073
                           found = 0:
02074
                          break;
02075
02076
                      if (found) {
02077
                       if (iz > 0 && iz < 140)
02078
                          met \rightarrow pt[ix][iy] = p2[iz];
02079
                        break;
02080
02081
                   }
02082
              }
02083
02084
         }
02085
02086
         /* Use dynamical tropopause... */
         else if (ctl->met_tropo == 5) {
02088
            /* Loop over grid points... */
for (ix = 0; ix < met->nx; ix++)
  for (iy = 0; iy < met->ny; iy++) {
02089
02090
02091
02092
02093
                 /* Interpolate potential vorticity profile... */
                 for (iz = 0; iz < met->np; iz++)
   pv[iz] = met->pv[ix][iy][iz];
02094
02095
                 gsl_spline_init(spline, z, pv, (size_t) met->np);
for (iz = 0; iz <= 160; iz++)</pre>
02096
02097
02098
                  pv2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02100
                 /* Interpolate potential temperature profile... */
                 for (iz = 0; iz < met->np; iz++)
  th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
02101
02102
                 gsl_spline_init(spline, z, th, (size_t) met->np); for (iz = 0; iz <= 160; iz++)
02103
02104
02105
                   th2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02106
02107
                 /\star Find dynamical tropopause 3.5 PVU + 380 K \star/
02108
                 met->pt[ix][iy] = GSL_NAN;
                 02109
02110
02111
                       met->pt[ix][iy] = p2[iz];
02112
02113
                     break;
02114
02115
              }
02116
02117
02118
02119
           ERRMSG("Cannot calculate tropopause!");
02120
02121
         gsl_spline_free(spline);
02122
02123
         gsl_interp_accel_free(acc);
02124 }
```

Here is the call graph for this function:



5.21.2.27 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 2128 of file libtrac.c.

```
02135
02136
02137
        FILE *in = NULL;
02138
        char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
02139
          msg[2 * LEN], rvarname[LEN], rval[LEN];
02140
02141
02142
02143
02144
         /* Open file... */
        if (filename[strlen(filename) - 1] != '-')
if (!(in = fopen(filename, "r")))
02145
02146
             ERRMSG("Cannot open file!");
02147
02149
         /* Set full variable name... */
        if (arridx >= 0) {
02150
         sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
02151
02152
02153
         sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
02154
02155
02156
02157
        /* Read data... */
02158
        if (in != NULL)
02159
         while (fgets(line, LEN, in))
02161
            if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
02162
               if (strcasecmp(rvarname, fullname1) == 0 |
02163
                    strcasecmp(rvarname, fullname2) == 0) {
02164
                 contain = 1;
02165
                 break;
02166
               }
        for (i = 1; i < argc - 1; i++)</pre>
        if (strcasecmp(argv[i], fullname1) == 0 ||
    strcasecmp(argv[i], fullname2) == 0) {
    sprintf(rval, "%s", argv[i + 1]);
02168
02169
02170
02171
            contain = 1;
02172
             break;
02173
02174
        /* Close file... */
if (in != NULL)
02175
02176
02177
          fclose(in);
02178
02179
        /* Check for missing variables... */
02180 if (!contain) {
         if (strlen(defvalue) > 0)
02181
02182
             sprintf(rval, "%s", defvalue);
02183
          else {
02184
             sprintf(msg, "Missing variable %s!\n", fullname1);
02185
             ERRMSG(msg);
02186
02187
02188
        /* Write info... */
02189
        printf("%s = %s\n", fullname1, rval);
02190
02192
         /* Return values... */
02193
        if (value != NULL)
02194
          sprintf(value, "%s", rval);
02195
         return atof(rval);
02196 }
```

5.21.2.28 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 2200 of file libtrac.c.

```
02209
02210
       struct tm t0, t1;
02211
       t0.tm_year = 100;
02212
02213
       t0.tm mon = 0;
02214
       t0.tm_mday = 1;
02215
       t0.tm\_hour = 0;
02216
       t0.tm_min = 0;
02217
       t0.tm_sec = 0;
02218
       t1.tm_year = year - 1900;
02219
       t1.tm_mon = mon - 1;
02220
02221
       t1.tm_mday = day;
02222
       t1.tm_hour = hour;
02223
       t1.tm_min = min;
       t1.tm_sec = sec;
02224
02225
       *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
02227 }
```

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

Measure wall-clock time.

Definition at line 2231 of file libtrac.c.

```
02234
                    {
02235
02236
         static double starttime[NTIMER], runtime[NTIMER];
02237
02238
        /* Check id... */
        if (id < 0 || id >= NTIMER)
02239
           ERRMSG("Too many timers!");
02241
02242
         /* Start timer... */
02243
        if (mode == 1) {
         if (starttime[id] <= 0)</pre>
02244
02245
             starttime[id] = omp_get_wtime();
02246
          else
02247
             ERRMSG("Timer already started!");
02248
02249
        /* Stop timer... */
else if (mode == 2) {
   if (starttime[id] > 0) {
02250
02251
02252
02253
             runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
02254
             starttime[id] = -1;
02255
02256 }
02257
02258
        /* Print timer... */
        else if (mode == 3) {
  printf("%s = %.3f s\n", name, runtime[id]);
  runtime[id] = 0.
02260
           runtime[id] = 0;
02261
02262
        }
02263 }
```

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

Write atmospheric data.

Definition at line 2267 of file libtrac.c.

```
02271 {
02272
02273 FILE *in, *out;
02274
02275 char line[LEN];
02276
02277 double r, t0, t1;
02278
02279 int ip, iq, year, mon, day, hour, min, sec;
02280
```

```
/* Set time interval for output... */
        t0 = t - 0.5 * ctl->dt_mod;

t1 = t + 0.5 * ctl->dt_mod;
02282
02283
02284
02285
        /* Write info... */
        printf("Write atmospheric data: %s\n", filename);
02286
02288
        /∗ Write ASCII data...
02289
        if (ctl->atm_type == 0) {
02290
          /* Check if gnuplot output is requested... */
if (ctl->atm_gpfile[0] != '-') {
02291
02292
02293
02294
             /* Create gnuplot pipe... */
             if (!(out = popen("gnuplot", "w")))
02295
               ERRMSG("Cannot create pipe to gnuplot!");
02296
02297
            /* Set plot filename... */
fprintf(out, "set out \"%s.png\"\n", filename);
02298
02300
02301
             /* Set time string... */
             jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02302
02303
02304
                      year, mon, day, hour, min);
02305
02306
             /* Dump gnuplot file to pipe... */
02307
             if (!(in = fopen(ctl->atm_gpfile, "r")))
              ERRMSG("Cannot open file!");
02308
             while (fgets(line, LEN, in))
fprintf(out, "%s", line);
02309
02310
02311
             fclose(in);
02312
          }
02313
02314
           else {
02315
             /* Create file... */
02316
             if (!(out = fopen(filename, "w")))
02317
               ERRMSG("Cannot create file!");
02318
02319
02320
02321
           /\star Write header... \star/
02322
          fprintf(out,
                    "# $1 = time [s]\n"
02323
                    "# $2 = altitude [km]\n"
02324
02325
                   "# $3 = longitude [deg] \n" "# <math>$4 = latitude [deg] \n");
          02326
02327
02328
02329
02330
02331
           /* Write data... */
02332
           for (ip = 0; ip < atm->np; ip++) {
02333
            /* Check time... */
if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
02334
02335
02336
               continue;
02337
02338
             /* Write output... */
            fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
02339
                     atm->lon[ip], atm->lat[ip]);
02340
             for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02341
02342
02343
02344
02345
             fprintf(out, "\n");
02346
          }
02347
02348
           /* Close file... */
02349
          fclose(out);
02350
02351
02352
        /* Write binary data... */
02353
        else if (ctl->atm_type == 1) {
02354
02355
           /* Create file... */
02356
          if (!(out = fopen(filename, "w")))
02357
             ERRMSG("Cannot create file!");
02358
02359
           /* Write data... */
          FWRITE(&atm->np, int,
02360
02361
                  1,
02362
                   out);
02363
           FWRITE(atm->time, double,
02364
                    (size_t) atm->np,
02365
                  out);
02366
          FWRITE(atm->p, double,
02367
                    (size_t) atm->np,
```

```
02368
                out);
02369
         FWRITE(atm->lon, double,
02370
                 (size_t) atm->np,
02371
                out);
         02372
02373
02374
                out);
02375
         for (iq = 0; iq < ctl->nq; iq++)
02376
         FWRITE(atm->q[iq], double,
02377
                   (size_t) atm->np,
02378
                 out);
02379
02380
         /* Close file... */
02381
         fclose(out);
02382
02383
       /* Error... */
02384
02385
       else
02386
         ERRMSG("Atmospheric data type not supported!");
02387 }
```

Here is the call graph for this function:



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

Write CSI data.

Definition at line 2391 of file libtrac.c.

```
02395
                    {
02396
02397
        static FILE *in, *out;
02398
02399
        static char line[LEN]:
02400
02401
        static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
02402
          rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
02403
02404
        static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
02405
02406
        /* Init... */
02407
        if (t == ctl->t_start) {
02408
02409
           /\star Check quantity index for mass... \star/
02410
           if (ctl->qnt_m < 0)
            ERRMSG("Need quantity mass!");
02411
02412
02413
           /* Open observation data file... */
          printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
02414
02415
             (!(in = fopen(ctl->csi_obsfile, "r")))
            ERRMSG("Cannot open file!");
02416
02417
          /* Create new file... */
printf("Write CSI data: %s\n", filename);
if (!(out = fopen(filename, "w")))
02418
02419
02420
02421
             ERRMSG("Cannot create file!");
02422
           /* Write header... */
02423
          fprintf(out, "# $1 = time [s]\n"
02424
02425
02426
                   "# $2 = number of hits (cx) \n"
02427
                    "# $3 = number of misses (cy) \n"
```

```
"# $4 = number of false alarms (cz)\n"
                    "# $5 = number of observations (cx + cy) \n"
02429
02430
                     "# $6 = number of forecasts (cx + cz)\n"
                     "# \$7 = bias (forecasts/observations) [\%] \n"
02431
                     "# $8 = probability of detection (POD) [%%]\n"
02432
                     "# $9 = false alarm rate (FAR) [%%]\n"
02433
02434
                     "# $10 = critical success index (CSI) [%%]\n\n");
02435
02436
        /* Set time interval... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
02437
02438
02439
02440
02441
         /* Initialize grid cells... */
02442 #pragma omp parallel for default(shared) private(ix,iy,iz)
02443
         for (ix = 0; ix < ctl->csi_nx; ix++)
           for (iy = 0; iy < ctl->csi_ny; iy++)
  for (iz = 0; iz < ctl->csi_nz; iz++)
    modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
02444
02445
02446
02447
         /* Read observation data... */
02448
02449
         while (fgets(line, LEN, in)) {
02450
           /* Read data... */
if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02451
02452
02453
02454
              continue;
02455
02456
           /* Check time... */
02457
           if (rt < t0)</pre>
02458
             continue;
02459
           if (rt > t1)
02460
             break;
02461
02462
           /* Calculate indices... */
           ix = (int) ((rlon - ctl->csi_lon0))
02463
           / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
iy = (int) ((rlat - ctl->csi_lat0)
02464
02465
02466
                           (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02467
           iz = (int) ((rz - ctl -> csi_z0)
02468
                         / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02469
           /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
02470
02471
                iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02472
02473
02474
02475
           /\star Get mean observation index... \star/
02476
           obsmean[ix][iy][iz] += robs;
02477
           obscount[ix][iy][iz]++;
02478
02479
         /* Analyze model data... */
02480
02481 \#pragma omp parallel for default(shared) private(ip,ix,iy,iz)
         for (ip = 0; ip < atm->np; ip++) {
02482
02483
02484
           /* Check time... */
02485
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
02486
             continue;
02487
           /* Get indices... */
ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
02488
02489
02490
                          / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
           iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
02491
02492
                          / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
           02493
02494
02495
02496
           /* Check indices... */
           if (ix < 0 || ix >= ctl->csi_nx ||
02497
02498
                iy < 0 \mid \mid iy >= ctl->csi_ny \mid \mid iz < 0 \mid \mid iz >= ctl->csi_nz)
02499
              continue;
02500
02501
           /\star Get total mass in grid cell... \star/
           modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02502
02503
02504
02505
         /* Analyze all grid cells... */
02506 #pragma omp parallel for default(shared) private(ix,iy,iz,dlon,dlat,lat,area)
02507 for (ix = 0; ix < ctl->csi_nx; ix++)
02508 for (iy = 0; iy < ctl->csi_ny; iy++)
02509 for (iz = 0; iz < ctl->csi_nz; iz++) {
02510
02511
                /* Calculate mean observation index... */
                if (obscount[ix][iy][iz] > 0)
  obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
02512
02513
02514
```

```
/* Calculate column density... */
                 if (modmean[ix][iy][iz] > 0) {
  dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
  dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
02516
02517
02518
                   lat = ctl->csi_lat1 - ctl->csi_lat0/, ctl->csi_ny,
lat = ctl->csi_lat0 + dlat * (iy + 0.5);
area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
* cos(lat * M_PI / 180.);
02519
02520
02521
02522
                    modmean[ix][iy][iz] /= (1e6 * area);
02523
02524
02525
                  /* Calculate CSI... */
02526
                 if (obscount[ix][iy][iz] > 0) {
02527
                   if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02528
                         modmean[ix][iy][iz] >= ctl->csi_modmin)
02529
                      cx++;
02530
                   else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
                                modmean[ix][iy][iz] < ctl->csi_modmin)
02531
                      cv++;
02532
02533
                    else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
                               modmean[ix][iy][iz] >= ctl->csi_modmin)
02534
02535
                      cz++;
02536
                 }
              }
02537
02538
02539
          /* Write output... */
         if (fmod(t, ctl->csi_dt_out) == 0) {
02540
02541
            02542
02543
                      t, cx, cy, cz, cx + cy, cx + cz, 

(cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN, 

(cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN, 

(cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
02544
02545
02546
02547
02548
                       (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
02549
            /* Set counters to zero... */
02550
02551
            cx = cy = cz = 0;
02553
02554
         /* Close file... */
02555
         if (t == ctl->t_stop)
02556
            fclose(out);
02557 }
```

5.21.2.32 void write ens (const char * filename, ctl t * ctl, atm t * atm, double t)

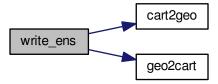
Write ensemble data.

Definition at line 2561 of file libtrac.c.

```
02565
                    {
02566
02567
        static FILE *out;
02568
02569
        static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
02570
          t0, t1, x[NENS][3], xm[3];
02571
02572
        static int ip, iq;
02573
02574
        static size t i, n:
02575
02576
        /* Init... */
02577
         if (t == ctl->t_start) {
02578
02579
          /* Check quantities... */
02580
          if (ctl->qnt_ens < 0)</pre>
02581
             ERRMSG("Missing ensemble IDs!");
02582
           /* Create new file... */
02583
           printf("Write ensemble data: %s\n", filename);
if (!(out = fopen(filename, "w")))
02584
02585
            ERRMSG("Cannot create file!");
02586
02587
           /* Write header... */
02588
           fprintf(out,
02589
02590
                    "# $1 = time [s] \n"
                    "# $2 = altitude [km] \n"
02591
                    "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02592
           for (iq = 0; iq < ctl->nq; iq+)
fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
02593
02594
```

```
ctl->qnt_name[iq], ctl->qnt_unit[iq]);
            02596
02597
02598
02599
02600
02601
02602
          /\star Set time interval... \star/
         t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
02603
02604
02605
02606
         /* Init...
02607
         ens = GSL_NAN;
02608
         n = 0;
02609
         /* Loop over air parcels... */
for (ip = 0; ip < atm->np; ip++) {
02610
02611
02612
02613
           /* Check time... */
02614
            if (atm->time[ip] < t0 || atm->time[ip] > t1)
02615
02616
02617
            /* Check ensemble id... */
            if (atm->q[ctl->qnt_ens][ip] != ens) {
02618
02619
02620
              /* Write results... */
02621
               if (n > 0) {
02622
02623
                 /* Get mean position... */
                xm[0] = xm[1] = xm[2] = 0;
for (i = 0; i < n; i++) {
  xm[0] += x[i][0] / (double) n;
  xm[1] += x[i][1] / (double) n;
  xm[2] += x[i][2] / (double) n;</pre>
02624
02625
02626
02627
02628
02629
                 cart2geo(xm, &dummy, &lon, &lat);
fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
02630
02631
02632
                           lat);
02633
02634
                 /* Get quantity statistics... */
                 for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02635
02636
02637
                   fprintf(out, ctl->qnt\_format[iq], gsl\_stats\_mean(q[iq], 1, n));\\
02638
                 for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02639
02640
02641
                   fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02642
02643
                 fprintf(out, " %lu\n", n);
02644
02645
02646
               /* Init new ensemble... */
02647
              ens = atm->q[ctl->qnt_ens][ip];
02648
              n = 0;
02649
02650
02651
            /* Save data... */
02652
            p[n] = atm->p[ip];
02653
            geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
02654
            for (iq = 0; iq < ctl->nq; iq++)
            q[iq][n] = atm->q[iq][ip];
if ((++n) >= NENS)
02655
02656
02657
              ERRMSG("Too many data points!");
02658
02659
02660
          /* Write results... */
02661
         if (n > 0) {
02662
02663
            /* Get mean position... */
            /* Get mean position... */
xm[0] = xm[1] = xm[2] = 0;
for (i = 0; i < n; i++) {
  xm[0] += x[i][0] / (double) n;
  xm[1] += x[i][1] / (double) n;
  xm[2] += x[i][2] / (double) n;
02664
02665
02666
02667
02668
02669
02670
            cart2geo(xm, &dummy, &lon, &lat);
02671
            fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
02672
02673
            /* Get quantity statistics... */
02674
            for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02675
02676
              fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02677
            for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02678
02679
02680
              fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02681
```

Here is the call graph for this function:



5.21.2.33 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 2692 of file libtrac.c.

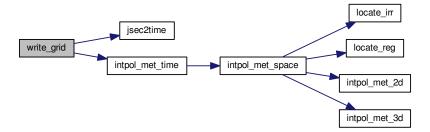
```
02698
                       {
02699
02700
          FILE *in, *out;
02701
02702
         char line[LEN];
02703
02704
         static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
02705
           area, rho_air, press, temp, cd, vmr, t0, t1, r;
02706
02707
         static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
02708
          /* Check dimensions... */
if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
    ERRMSG("Grid dimensions too large!");
02709
02710
02711
02712
02713
          /* Check quantity index for mass... */
02714
         if (ctl->qnt_m < 0)
02715
            ERRMSG("Need quantity mass!");
02716
         /* Set time interval for output... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
02717
02718
02719
02720
02721
          /* Set grid box size... */
02722
          dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
         dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
02723
02724
02725
02726
          /* Initialize grid... */
02727 #pragma omp parallel for default(shared) private(ix,iy,iz)
02728
         for (ix = 0; ix < ctl->grid_nx; ix++)
            for (iy = 0; iy < ctl->grid_ny; iy++)
  for (iz = 0; iz < ctl->grid_nz; iz++)
   mass[ix][iy][iz] = 0;
02729
02730
02731
02732
02733
          /* Average data... */
02734 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
02735 for (ip = 0; ip < atm->np; ip++)
02736
           if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {</pre>
02737
02738
               /* Get index... */
02739
               ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
```

```
iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
02741
02742
               /* Check indices... */
if (ix < 0 || ix >= ctl->grid_nx ||
    iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
02743
02744
02745
02746
02747
               /* Add mass... */
02748
              mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02749
02750
02751
         /* Check if gnuplot output is requested... */
if (ctl->grid_gpfile[0] != '-') {
02752
02753
02754
02755
            /* Write info... */
            printf("Plot grid data: %s.png\n", filename);
02756
02757
            /* Create gnuplot pipe... */
            if (!(out = popen("gnuplot", "w")))
02759
02760
               ERRMSG("Cannot create pipe to gnuplot!");
02761
            /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
02762
02763
02764
02765
            /* Set time string... */
            jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02766
02767
02768
                      year, mon, day, hour, min);
02769
            /* Dump gnuplot file to pipe... */
if (!(in = fopen(ctl->grid_gpfile, "r")))
    ERRMSG("Cannot open file!");
02770
02772
            while (fgets(line, LEN, in))
fprintf(out, "%s", line);
02773
02774
02775
            fclose(in);
02776
02777
02778
         else {
02779
02780
            /* Write info... */
02781
           printf("Write grid data: %s\n", filename);
02782
02783
            /* Create file... */
02784
           if (!(out = fopen(filename, "w")))
02785
               ERRMSG("Cannot create file!");
02786
02787
02788
         /* Write header... */
02789
         fprintf(out,
                    "# $1 = time [s]\n"
02791
                    "# $2 = altitude [km] \n"
02792
                    "# $3 = longitude [deg] \n"
                    "# $4 = latitude [deg]\n"
02793
                    "# $5 = surface area [km^2]\n"
02794
                    "# $6 = layer width [km]\n"
"# $7 = temperature [K]\n"
02795
02796
02797
                    "# $8 = \text{column density } [kg/m^2] \n"
02798
                    "# $9 = volume mixing ratio [1] n n");
02799
         /* Write data... */
02800
         for (ix = 0; ix < ctl->grid_nx; ix++) {
   if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
02801
02802
               fprintf(out, "\n");
02803
02804
                 (iy = 0; iy < ctl->grid_ny; iy++) {
              if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
    fprintf(out, "\n");
for (iz = 0; iz < ctl->grid_nz; iz++)
02805
02806
02807
02808
                 if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
02810
                    /* Set coordinates... */
02811
                    z = ctl->grid_z0 + dz * (iz + 0.5);
                    lon = ctl->grid_lon0 + dlon * (ix + 0.5);
lat = ctl->grid_lat0 + dlat * (iy + 0.5);
02812
02813
02814
02815
                    /* Get pressure and temperature... */
02816
                    press = P(z);
02817
                    intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
                                        NULL, &temp, NULL, NULL, NULL, NULL, NULL, NULL);
02818
02819
                    /* Calculate surface area... */
area = dlat * dlon * SQR(RE * M_PI / 180.)
02820
02821
02822
                      * cos(lat * M_PI / 180.);
02823
                    /* Calculate column density... */
cd = mass[ix][iy][iz] / (1e6 * area);
02824
02825
02826
```

```
/* Calculate volume mixing ratio... */
               rho_air = 100. * press / (RA * temp);

vmr = MA / ctl->molmass * mass[ix][iy][iz]
02828
02829
                 / (rho_air * 1e6 * area * 1e3 * dz);
02830
02831
               02832
02834
02835
02836
         }
       }
02837
02838
02839
        /* Close file... */
02840
       fclose(out);
02841 }
```

Here is the call graph for this function:



5.21.2.34 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 2845 of file libtrac.c.

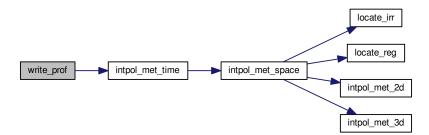
```
02851
02852
02853
         static FILE *in, *out;
02854
02855
        static char line[LEN];
02856
02857
         static double mass[GX][GY][GZ], obsmean[GX][GY], obsmean2[GX][GY], rt, rz,
02858
           rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp,
02859
          rho_air, vmr, h2o, o3;
02860
02861
         static int obscount[GX][GY], ip, ix, iv, iz, okav;
02862
02863
         /* Init... */
02864
         if (t == ctl->t_start) {
02865
           /* Check quantity index for mass... */
if (ctl->qnt_m < 0)</pre>
02866
02867
             ERRMSG("Need quantity mass!");
02868
02869
02870
           /* Check dimensions... */
           if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
    ERRMSG("Grid dimensions too large!");
02871
02872
02873
02874
           /* Open observation data file... */
02875
           printf("Read profile observation data: %s\n", ctl->prof_obsfile);
02876
           if (!(in = fopen(ctl->prof_obsfile, "r")))
02877
             ERRMSG("Cannot open file!");
02878
           /* Create new output file... */
printf("Write profile data: %s\n", filename);
if (!(out = fopen(filename, "w")))
02879
02880
02881
02882
             ERRMSG("Cannot create file!");
```

```
/* Write header... */
02884
02885
            fprintf(out,
                      "# $1 = time [s] \n"
02886
                      "# $2 = altitude [km] \n"
02887
                      "# $3 = longitude [deg]\n"
02888
                      "# $4 = latitude [deg]\n"
02890
                      "# $5 = pressure [hPa] \n"
02891
                      "# $6 = temperature [K] \n"
                      "# \$7 = volume mixing ratio [1] \n"
02892
                      "# $8 = H20 volume mixing ratio [1]\n"
02893
                     "# $9 = O3 volume mixing ratio [1]\n"
"# $10 = observed BT index (mean) [K]\n"
02894
02895
02896
                     "# $11 = observed BT index (sigma) [K]\n");
02897
            /\star Set grid box size... \star/
02898
           dz = (ctl->prof_zl - ctl->prof_z0) / ctl->prof_nz;
dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
02899
02900
02901
02902
02903
02904
         /* Set time interval... */
         t0 = t - 0.5 * ctl->dt_mod;

t1 = t + 0.5 * ctl->dt_mod;
02905
02906
02907
02908
          /* Initialize... */
02909 #pragma omp parallel for default(shared) private(ix,iy,iz)
02910
         for (ix = 0; ix < ctl->prof_nx; ix++)
02911
           for (iy = 0; iy < ctl->prof_ny; iy++) {
              obsmean[ix][iy] = 0;
02912
02913
              obsmean2[ix][iy] = 0;
02914
              obscount[ix][iy] = 0;
02915
              for (iz = 0; iz < ctl->prof_nz; iz++)
02916
                mass[ix][iy][iz] = 0;
02917
02918
         /* Read observation data... */
02919
         while (fgets(line, LEN, in)) {
02920
02921
            /* Read data... */
if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02922
02923
02924
                 5)
02925
              continue:
02926
02927
            /* Check time... */
02928
            if (rt < t0)
02929
              continue;
            if (rt > t1)
02930
02931
              break:
02932
02933
            /* Calculate indices... */
           ix = (int) ((rlon - ctl->prof_lon0) / dlon);
iy = (int) ((rlat - ctl->prof_lat0) / dlat);
02934
02935
02936
            /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
02937
02938
              continue;
02940
02941
            /* Get mean observation index... */
            obsmean[ix][iy] += robs;
obsmean2[ix][iy] += SQR(robs);
02942
02943
02944
            obscount[ix][iy]++;
02945
02946
02947
         /* Analyze model data... */
02948 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
02949 for (ip = 0; ip < atm->np; ip++) {
02950
02951
            /* Check time... */
            if (atm->time[ip] < t0 || atm->time[ip] > t1)
02953
02954
            /* Get indices... */
02955
           ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
02956
02957
02958
            iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
02959
            /* Check indices... */
02960
            if (ix < 0 || ix >= ctl->prof_nx ||
   iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
02961
02962
02963
              continue;
02964
02965
            /* Get total mass in grid cell... */
02966
            mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02967
02968
02969
         /* Extract profiles... */
```

```
for (ix = 0; ix < ctl->prof_nx; ix++)
          for (iy = 0; iy < ctl->prof_ny; iy++)
    if (obscount[ix][iy] > 0) {
02971
02972
02973
               /* Check profile... */
02974
               okay = 0;
for (iz = 0; iz < ctl->prof_nz; iz++)
  if (mass[ix][iy][iz] > 0) {
02975
02976
02977
02978
                   okay = 1;
02979
                    break;
02980
02981
                if (!okav)
02982
                 continue;
02983
02984
                /∗ Write output...
02985
                fprintf(out, "\n");
02986
               /* Loop over altitudes... */
for (iz = 0; iz < ctl->prof_nz; iz++) {
02987
02988
02990
                  /* Set coordinates... */
                 z = ctl->prof_z0 + dz * (iz + 0.5);
lon = ctl->prof_lon0 + dlon * (ix + 0.5);
02991
02992
                  lat = ctl->prof_lat0 + dlat * (iy + 0.5);
02993
02994
02995
                  /\star Get pressure and temperature... \star/
02996
02997
                  intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
02998
                                    NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
02999
                 /* Calculate surface area... */
area = dlat * dlon * SQR(M_PI * RE / 180.)
    * cos(lat * M_PI / 180.);
03000
03001
03002
03003
03004
                  /\star Calculate volume mixing ratio... \star/
                  rho_air = 100. * press / (RA * temp);
vmr = MA / ctl->molmass * mass[ix][iy][iz]
03005
03006
                    / (rho_air * area * dz * 1e9);
03007
03008
03009
                  /* Write output... */
                  03010
0.3011
03012
                           03013
03014
03015
03016
             }
03017
         /* Close file... */
03018
        if (t == ctl->t_stop)
03019
03020
           fclose(out);
03021 }
```

Here is the call graph for this function:



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

Write station data.

Definition at line 3025 of file libtrac.c.

```
03029
                   {
03030
03031
        static FILE *out;
03032
03033
        static double rmax2, t0, t1, x0[3], x1[3];
03034
        static int ip, iq;
03036
        /* Init... */
03037
03038
        if (t == ctl->t_start) {
03039
          /* Write info... */
03040
          printf("Write station data: %s\n", filename);
03041
03042
03043
          /\star Create new file... \star/
          if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");
03044
03045
03046
03047
          /* Write header... */
03048
          fprintf(out,
03049
                   "# $1 = time [s] \n"
                   "# $2 = altitude [km] \n"
03050
                   "# $3 = longitude [deg] \n" "# $4 = latitude [deg] \n");
03051
          03052
03053
03054
03055
          fprintf(out, "\n");
03056
03057
          /\star Set geolocation and search radius... \star/
03058
          geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
03059
          rmax2 = SOR(ctl->stat r);
03060
03061
03062
        /\star Set time interval for output... \star/
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
03063
03064
03065
03066
        /* Loop over air parcels... */
03067
        for (ip = 0; ip < atm->np; ip++) {
03068
          /* Check time... */
if (atm->time[ip] < t0 || atm->time[ip] > t1)
03069
03070
03071
            continue;
03072
03073
          /* Check station flag... */
03074
          if (ctl->qnt_stat >= 0)
03075
           if (atm->q[ctl->qnt_stat][ip])
03076
              continue;
03077
03078
          /* Get Cartesian coordinates... */
03079
          geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
03080
03081
          /\star Check horizontal distance... \star/
03082
          if (DIST2(x0, x1) > rmax2)
03083
            continue;
03084
03085
          /* Set station flag... */
03086
          if (ctl->qnt_stat >= 0)
03087
            atm->q[ctl->qnt_stat][ip] = 1;
03088
          /* Write data... */
fprintf(out, "%.2f %g %g %g",
03089
03090
03091
                  atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
           for (iq = 0; iq < ctl->nq; iq++) {
   fprintf(out, " ");
03092
03093
03094
            fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
03095
03096
          fprintf(out, "\n");
03097
03098
03099
        /* Close file... */
0.3100
        if (t == ctl->t_stop)
0.3101
          fclose(out);
03102 }
```

Here is the call graph for this function:



5.22 libtrac.h

```
00001 /*
00002
        This file is part of MPTRAC.
00003
00004
        \ensuremath{\mathsf{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.
80000
00009
        \ensuremath{\mathsf{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 <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>.
00016
        Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00017
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 /*
08000
00081
00082
00084 #define LEN 5000
00085
00087 #define NP 10000000
00088
```

5.22 libtrac.h 179

```
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 /* --
        Macros...
00118
00119
00121 #define ALLOC(ptr, type, n)
00122 if((ptr=calloc((size_t)(n), sizeof(type)))==NULL)
         ERRMSG("Out of memory!");
00123
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
        : (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)
00151 #define DIST(a, b) sqrt(DIST2(a, b))
00152
00154 #define DIST2(a, b)
       ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))
00155
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, 1%d): %s\n\n",
         __FILE__, __func__, __LINE__, msg);
exit(EXIT_FAILURE);
00163
00164
00165
00166
00168 #define FREAD(ptr, type, size, out) {
00169    if(fread(ptr, sizeof(type), size, out)!=size)
00170    ERRMSG("Error while reading!");
00171
00172
00176
            ERRMSG("Error while writing!");
00177
00178
00180 #define LIN(x0, y0, x1, y1, x)
00181 ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))
00182
00184 #define NC(cmd) {
       if((cmd)!=NC_NOERR)
00185
00186
           ERRMSG(nc_strerror(cmd));
00187
00188
00190 #define NORM(a) sqrt(DOTP(a, a))
00191
00193 #define PRINT(format, var)
00194 printf("Print (%s, %s, 1%d): %s= "format"\n",
00195 __FILE__, _func__, _LINE__, #var, var);
00196
00198 #define P(z) (P0*exp(-(z)/H0))
00199
00201 #define SQR(x) ((x) *(x))
00202
00204 #define THETA(p, t) ((t)*pow(1000./(p), 0.286))
```

```
00211
00212
00214 #define Z(p) (H0*log(P0/(p)))
00215
00216 /* -----
        Timers...
00217
00218
00219
00221 #define START_TIMER(id) timer(#id, id, 1)
00222
00224 #define STOP_TIMER(id) timer(#id, id, 2)
00225
00227 #define PRINT_TIMER(id) timer(#id, id, 3)
00228
00230 #define NTIMER 12
00231
00233 #define TIMER_TOTAL 0
00234
00236 #define TIMER INIT 1
00237
00239 #define TIMER_INPUT 2
00240
00242 #define TIMER_OUTPUT 3
00243
00245 #define TIMER ADVECT 4
00246
00248 #define TIMER_DECAY 5
00249
00251 #define TIMER_DIFFMESO 6
00252
00254 #define TIMER_DIFFTURB 7
00255
00257 #define TIMER_ISOSURF 8
00258
00260 #define TIMER_METEO 9
00261
00263 #define TIMER POSITION 10
00264
00266 #define TIMER_SEDI 11
00267
00268 /* ---
00269
        Structs...
00270
00271
00273 typedef struct {
00274
00276
00277
00279
       char qnt_name[NQ][LEN];
00280
00282
       char gnt unit[NO][LEN];
00283
00285
       char qnt_format[NQ][LEN];
00286
00288
       int qnt_ens;
00289
00291
       int qnt_m;
00292
00294
       int qnt_rho;
00295
00297
       int qnt_r;
00298
00300
       int qnt_ps;
00301
00303
       int qnt_pt;
00304
00306
       int qnt_z;
00307
00309
       int qnt_p;
00310
       int qnt_t;
00313
00315
       int qnt_u;
00316
00318
       int gnt v;
00319
00321
       int qnt_w;
00322
00324
       int qnt_h2o;
00325
00327
       int qnt_o3;
00328
```

5.22 libtrac.h 181

```
int qnt_theta;
00331
00333
        int qnt_vh;
00334
00336
        int qnt_vz;
00337
        int qnt_pv;
00340
00342
        int qnt_tice;
00343
00345
        int qnt_tsts;
00346
00348
        int gnt tnat;
00349
00351
        int qnt_stat;
00352
        int direction;
00354
00355
00357
        double t_start;
00358
00360
        double t_stop;
00361
00363
        double dt_mod;
00364
00366
        double dt_met;
00367
00369
        int met_dx;
00370
00372
        int met_dy;
00373
00375
        int met_dp;
00376
00378
        int met_sx;
00379
00381
        int met_sy;
00382
00384
        int met_sp;
00385
00387
        int met_np;
00388
00390
        double met_p[EP];
00391
00394
        int met_tropo;
00395
        char met_geopot[LEN];
00398
00400
        double met_dt_out;
00401
00403
        char met_stage[LEN];
00404
00407
        int isosurf;
00408
00410
        char balloon[LEN];
00411
00413
        double turb_dx_trop;
00414
        double turb_dx_strat;
00417
00419
        double turb_dz_trop;
00420
        double turb_dz_strat;
00422
00423
00425
        double turb_mesox;
00426
00428
        double turb_mesoz;
00429
00431
        double molmass;
00432
00434
        double tdec trop:
00435
00437
        double tdec_strat;
00438
00440
        double psc_h2o;
00441
00443
        double psc_hno3;
00444
00446
        char atm_basename[LEN];
00447
00449
        char atm_gpfile[LEN];
00450
00452
        double atm_dt_out;
00453
00455
        int atm_filter;
00456
00458
        int atm_type;
00459
        char csi basename[LEN];
00461
```

```
00462
00464
        double csi_dt_out;
00465
00467
        char csi_obsfile[LEN];
00468
00470
        double csi_obsmin;
00471
00473
        double csi_modmin;
00474
00476
        int csi_nz;
00477
00479
        double csi_z0;
00480
00482
        double csi_z1;
00483
00485
        int csi_nx;
00486
00488
        double csi lon0;
00489
00491
        double csi_lon1;
00492
00494
        int csi_ny;
00495
00497
        double csi_lat0;
00498
00500
        double csi_lat1;
00501
00503
        char grid_basename[LEN];
00504
        char grid_gpfile[LEN];
00506
00507
00509
        double grid_dt_out;
00510
00512
        int grid_sparse;
00513
00515
        int grid_nz;
00516
        double grid_z0;
00519
00521
        double grid_z1;
00522
00524
        int grid_nx;
00525
00527
        double grid_lon0;
00528
00530
        double grid_lon1;
00531
00533
        int grid_ny;
00534
00536
        double grid_lat0;
00537
00539
        double grid_lat1;
00540
00542
        char prof_basename[LEN];
00543
00545
        char prof_obsfile[LEN];
00546
00548
        int prof_nz;
00549
00551
        double prof_z0;
00552
00554
        double prof_z1;
00555
00557
        int prof_nx;
00558
00560
        double prof_lon0;
00561
00563
        double prof_lon1;
00564
        int prof_ny;
00567
00569
        double prof_lat0;
00570
00572
        double prof_lat1;
00573
00575
        char ens_basename[LEN];
00576
00578
00579
        char stat_basename[LEN];
00581
        double stat lon;
00582
        double stat_lat;
00585
00587
        double stat_r;
00588
00589 }
       ctl_t;
00590
```

5.22 libtrac.h 183

```
00592 typedef struct {
00593
00595
        int np;
00596
00598
        double time[NP];
00599
00601
        double p[NP];
00602
00604
        double lon[NP];
00605
        double lat[NP];
00607
00608
00610
        double q[NQ][NP];
00611
00613
        float up[NP];
00614
00616
        float vp[NP];
00617
00619
        float wp[NP];
00620
00622
        double cache_time[EX][EY][EP];
00623
00625
        float cache_usig[EX][EY][EP];
00626
00628
        float cache_vsig[EX][EY][EP];
00629
00631
        float cache_wsig[EX][EY][EP];
00632
00633 } atm_t;
00634
00636 typedef struct {
00637
00639
        double time;
00640
00642
        int nx;
00643
00645
        int ny;
00646
00648
        int np;
00649
00651
        double lon[EX];
00652
00654
        double lat[EY];
00655
00657
        double p[EP];
00658
00660
        double ps[EX][EY];
00661
        double pt[EX][EY];
00663
00664
00666
        float z[EX][EY][EP];
00667
00669
        float t[EX][EY][EP];
00670
00672
        float u[EX][EY][EP];
00673
00675
        float v[EX][EY][EP];
00676
00678
        float w[EX][EY][EP];
00679
00681
        float pv[EX][EY][EP];
00682
00684
        float h2o[EX][EY][EP];
00685
00687
        float o3[EX][EY][EP];
00688
00690
        float pl[EX][EY][EP];
00691
00692 } met_t;
00693
00694 /* -----
00695
         Functions...
00696
00697
00699 void cart2geo(
00700
       double *x,
00701
        double *z,
00702
        double *lon,
00703
        double *lat);
00704
00706 double clim_hno3(
00707
        double t,
00708
        double lat,
00709
        double p);
00710
00712 double clim_tropo(
00713
        double t.
```

```
00714
        double lat);
00715
00717 void day2doy(
00718
        int year,
00719
        int mon,
00720
        int day,
00721
        int *doy);
00722
00724 void doy2day(
00725
        int year,
00726
        int doy,
00727
        int *mon,
        int *day);
00728
00729
00731 void geo2cart(
        double z,
double lon,
00732
00733
00734
        double lat,
00735
        double *x);
00736
00738 void get_met(
00739
        ctl_t * ctl,
00740
        char *metbase,
00741
        double t,
met_t ** met0,
met_t ** met1);
00742
00743
00744
00746 void get_met_help(
00747
        double t,
00748
        int direct,
00749
        char *metbase.
00750
        double dt_met,
00751
        char *filename);
00752
00754 void intpol_met_2d(
00755
        double array[EX][EY],
00756
        int ix,
        int iy,
00757
00758
        double wx,
00759
        double wy,
00760
        double *var);
00761
00763 void intpol_met_3d(
00764 float array[EX][EY][EP],
00765
        int ip,
00766
        int ix,
00767
        int iy,
00768
        double wp,
00769
        double wx.
00770
        double wy,
00771
        double *var);
00772
00774 void intpol_met_space(
        met_t * met,
double p,
double lon,
00775
00776
00777
00778
        double lat,
00779
        double *ps,
00780
        double *pt,
00781
        double *z,
00782
        double *t,
00783
        double *u,
00784
        double *v,
00785
        double *w,
00786
        double *pv,
00787
        double *h2o,
        double *o3);
00788
00789
00791 void intpol_met_time(
        met_t * met0,
met_t * met1,
00792
00793
00794
        double ts,
        double p, double lon,
00795
00796
00797
        double lat,
00798
        double *ps,
00799
        double *pt,
00800
        double *z,
00801
        double *t,
00802
        double *11.
00803
        double *v,
00804
        double *w,
00805
        double *pv,
00806
        double *h2o,
00807
        double *o3);
00808
00810 void jsec2time(
```

5.22 libtrac.h 185

```
00811
        double jsec,
00812
        int *year,
00813
        int *mon,
00814
        int *day,
00815
        int *hour,
00816
        int *min.
        int *sec,
00818
        double *remain);
00819
00821 int locate_irr(
        double *xx,
00822
00823
        int n.
00824
        double x);
00825
00827 int locate_reg(
00828
       double *xx,
00829
        int n,
00830
        double x);
00831
00833 void read_atm(
00834
       const char *filename,
00835
        ctl_t * ctl,
        atm_t * atm);
00836
00837
00839 void read_ctl(
       const char *filename,
        int argc,
00841
00842
        char *argv[],
00843
        ctl_t * ctl);
00844
00846 void read_met(
00847
        ctl_t * ctl,
00848
        char *filename,
00849
        met_t * met);
00850
00852 void read_met_extrapolate(
00853
        met_t * met);
00856 void read_met_geopot(
       ctl_t * ctl,
met_t * met);
00857
00858
00859
00861 void read_met_help(
00862
        int ncid,
        char *varname,
00864
        char *varname2,
00865
        met_t * met,
00866
        float dest[EX][EY][EP],
00867
        float scl);
00868
00870 void read_met_ml2pl(
00871 ctl_t * ctl,
00872 met_t * met,
00873
        float var[EX][EY][EP]);
00874
00876 void read_met_periodic(
00877
        met_t * met);
00878
00880 void read_met_pv(
00881
        met_t * met);
00882
00884 void read_met_sample(
00885 ctl_t * ctl,
00886
        met_t * met);
00887
00889 void read_met_tropo(
00890
        ctl_t * ctl,
met_t * met);
00891
00892
00894 double scan_ctl(
00895
        const char *filename,
00896
        int argc,
        char *argv[],
const char *varname,
00897
00898
00899
        int arridx,
00900
        const char *defvalue,
00901
        char *value);
00902
00904 void time2jsec(
00905
        int year,
00906
        int mon,
00907
        int day,
00908
        int hour,
00909
        int min,
00910
        int sec,
        double remain,
double *jsec);
00911
00912
```

```
00915 void timer(
00916
       const char *name,
00917
        int id,
00918
        int mode);
00919
00921 void write_atm(
00922 const char *filename,
        ctl_t * ctl,
atm_t * atm,
00923
00924
00925
       double t);
00926
00928 void write_csi(
00929 const char *filename,
        ctl_t * ctl,
atm_t * atm,
00930
00931
00932
        double t);
00933
00935 void write_ens(
       const char *filename,
        ctl_t * ctl,
atm_t * atm,
00937
00938
00939
        double t);
00940
00942 void write_grid(
00943 const char *filename,
        ctl_t * ctl,
met_t * met0,
00944
00945
        met_t * met1,
atm_t * atm,
00946
00947
00948
        double t);
00949
00951 void write_prof(
00952
       const char *filename,
        ctl_t * ctl,
met_t * met0,
00953
00954
        met_t * met1,
atm_t * atm,
00955
00956
00957
        double t);
00958
00960 void write_station(
00961 const char *filename,
        ctl_t * ctl,
atm_t * atm,
00962
00963
00964
       double t);
```

5.23 met_map.c File Reference

Extract global map from meteorological data.

Functions

• int main (int argc, char *argv[])

5.23.1 Detailed Description

Extract global map from meteorological data.

Definition in file met_map.c.

5.23.2 Function Documentation

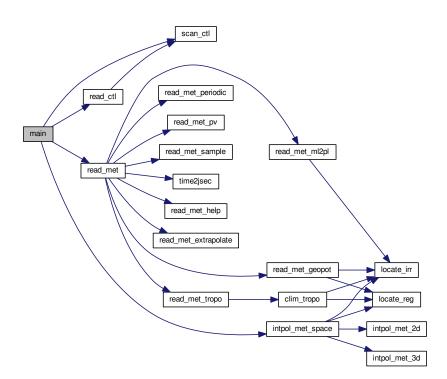
5.23.2.1 int main (int argc, char * argv[])

Definition at line 27 of file met_map.c.

```
00029
00030
00031
         ctl_t ctl;
00032
00033
         met t *met;
00034
         FILE *out;
00036
         static double dz, dzmin = 1e10, z, timem[EX][EY], psm[EX][EY], ptm[EX][EY], tm[EX][EY], um[EX][EY], vm[EX][EY], wm[EX][EY], h2om[EX][EY], o3m[EX][EY], zm[EX][EY], pvm[EX][EY], zt, ztm[EX][EY], tt, ttm[EX][EY];
00037
00038
00039
00040
00041
         static int i, ip, ip2, ix, iy, np[EX][EY];
00042
00043
         /* Allocate... */
00044
         ALLOC(met, met_t, 1);
00045
00046
         /* Check arguments... */
00047
         if (argc < 4)
00048
           ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00049
00050
         /* Read control parameters... */
00051
         read_ctl(argv[1], argc, argv, &ctl);
         z = scan\_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00052
00053
00054
         /* Loop over files... */
00055
         for (i = 3; i < argc; i++) {</pre>
00056
00057
           /* Read meteorological data... */
00058
           read_met(&ctl, argv[i], met);
00059
00060
           /* Find nearest pressure level... */
00061
           for (ip2 = 0; ip2 < met->np; ip2++) {
00062
             dz = fabs(Z(met->p[ip2]) - z);
             if (dz < dzmin) {
  dzmin = dz;</pre>
00063
00064
00065
               ip = ip2;
00066
00067
00068
00069
           /* Average data... */
           for (ix = 0; ix < met->nx; ix++)
00070
00071
            for (iy = 0; iy < met->ny; iy++) {
00072
               intpol_met_space(met, met->pt[ix][iy], met->lon[ix], met->
      lat[iy],
00073
                                    NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL,
00074
                                   NULL);
               timem[ix][iy] += met->time;
00075
                zm[ix][iy] += met->z[ix][iy][ip];
tm[ix][iy] += met->t[ix][iy][ip];
00076
00077
                um[ix][iy] += met->u[ix][iy][ip];
00079
                vm[ix][iy] += met->v[ix][iy][ip];
00080
                wm[ix][iy] += met->w[ix][iy][ip];
               pvm[ix][iy] += met->pv[ix][iy][ip];
h2om[ix][iy] += met->h2o[ix][iy][ip];
00081
00082
               o3m[ix][iy] += met->o3[ix][iy][ip];
psm[ix][iy] += met->ps[ix][iy];
00083
00085
               ptm[ix][iy] += met->pt[ix][iy];
00086
                ztm[ix][iy] += zt;
00087
                ttm[ix][iy] += tt;
00088
               np[ix][iy]++;
00089
00090
         }
00091
00092
         /* Create output file... */
00093
         printf("Write meteorological data file: %s\n", argv[2]);
00094
         if (!(out = fopen(argv[2], "w")))
00095
           ERRMSG("Cannot create file!");
00096
00097
         /* Write header... */
00098
         fprintf(out,
00099
                  "# $1
                          = time [s]\n"
                  "# $2 = altitude [km]\n"
00100
                   "# $3 = longitude [deg] \n"
00101
                  "# $4 = latitude [deg]\n"
"# $5 = pressure [hPa]\n"
00102
00103
00104
                  "# $6 = temperature [K]\n"
00105
                  "# $7 = zonal wind [m/s]\n"
                  "# $8 = meridional wind [m/s]\n"
00106
                  "# $9 = vertical wind [hPa/s]\n"
00107
                  "# $10 = H20 volume mixing ratio [1]\n");
00108
00109
        fprintf(out,
                  "# $11 = 03 volume mixing ratio [1]\n"
00110
00111
                  "# $12 = geopotential height [km]\n"
                  "# $13 = potential vorticity [PVU]\n"
"# $14 = surface pressure [hPa]\n"
00112
00113
                  "# $15 = tropopause pressure [hPa]\n"
00114
```

```
"# $16 = tropopause geopotential height [km]\n"
00116
                        "# $17 = tropopause temperature [K]\n");
00117
            /* Write data... */
for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
  for (ix = 0; ix < met->nx; ix++)
00118
00119
00120
00121
                  00122
00123
00124
00125
00126
00127
00128
                                 zm[ix][iy] / np[ix][iy], pvm[ix][iy] / np[ix][iy],
psm[ix][iy] / np[ix][iy], ptm[ix][iy] / np[ix][iy],
ztm[ix][iy] / np[ix][iy], ttm[ix][iy] / np[ix][iy]);
00129
00130
00131
               for (ix = 0; ix < met\rightarrownx; ix++)
00132
                 00133
00134
00135
00136
                                met->lon(ix), met->lat(iy), met->p(ip),
tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
tm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
zm[ix][iy] / np[ix][iy], pvm[ix][iy] / np[ix][iy],
psm[ix][iy] / np[ix][iy], ptm[ix][iy] / np[ix][iy],
ztm[ix][iy] / np[ix][iy], ttm[ix][iy] / np[ix][iy]);
00137
00138
00139
00140
00141
00142
00143
00144
00145
            /* Close file... */
00146
           fclose(out);
00147
00148
           /* Free... */
00149
           free(met);
00150
00151
            return EXIT SUCCESS;
00152 }
```

Here is the call graph for this function:



5.24 met map.c 189

5.24 met_map.c

```
00001 /*
        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
        the Free Software Foundation, either version 3 of the License, or
00006
00007
        (at your option) any later version.
80000
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
        You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
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
        met_t *met;
00034
00035
        FILE *out;
00036
        static double dz, dzmin = 1e10, z, timem[EX][EY], psm[EX][EY], ptm[EX][EY],
tm[EX][EY], um[EX][EY], vm[EX][EY], wm[EX][EY], h2om[EX][EY], o3m[EX][EY],
00037
00038
00039
           zm[EX][EY], pvm[EX][EY], zt, ztm[EX][EY], tt, ttm[EX][EY];
00040
00041
        static int i, ip, ip2, ix, iy, np[EX][EY];
00042
00043
         /* Allocate... */
00044
        ALLOC(met, met_t, 1);
00045
00046
        /* Check arguments... */
00047
        if (argc < 4)
00048
          ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00049
00050
        /* Read control parameters... */
        read_ct1(argv[1], argc, argv, &ct1);
z = scan_ct1(argv[1], argc, argv, "Z", -1, "", NULL);
00051
00052
00053
00054
         /* Loop over files... */
00055
        for (i = 3; i < argc; i++) {</pre>
00056
00057
           /* Read meteorological data... */
00058
           read_met(&ctl, argv[i], met);
00059
00060
           /* Find nearest pressure level... */
           for (ip2 = 0; ip2 < met->np; ip2++) {
  dz = fabs(Z(met->p[ip2]) - z);
00061
00062
00063
             if (dz < dzmin) {
00064
               dzmin = dz;
00065
               ip = ip2;
00066
            }
00067
          }
00068
00069
           /* Average data... */
           for (ix = 0; ix < met->nx; ix++)
00070
00071
            for (iy = 0; iy < met->ny; iy++) {
00072
               intpol_met_space(met, met->pt[ix][iy], met->lon[ix], met->
      lat[iy],
00073
                                   NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL,
00074
                                  NULL);
               timem[ix][iy] += met->time;
00075
00076
                zm[ix][iy] += met->z[ix][iy][ip];
00077
                tm[ix][iy] += met->t[ix][iy][ip];
00078
               um[ix][iy] += met->u[ix][iy][ip];
00079
               vm[ix][iy] += met->v[ix][iy][ip];
               wm[ix][iy] += met->w[ix][iy][ip];
00080
00081
               pvm[ix][iy] += met->pv[ix][iy][ip];
00082
               h2om[ix][iy] += met->h2o[ix][iy][ip];
00083
               o3m[ix][iy] += met->o3[ix][iy][ip];
00084
               psm[ix][iy] += met->ps[ix][iy];
               ptm[ix][iy] += met->pt[ix][iy];
00085
00086
               ztm[ix][iy] += zt;
00087
               ttm[ix][iy] += tt;
00088
               np[ix][iy]++;
```

```
00089
             }
00090
00091
00092
        /* Create output file... */
        printf("Write meteorological data file: %s\n", argv[2]);
if (!(out = fopen(argv[2], "w")))
00093
00094
          ERRMSG("Cannot create file!");
00096
00097
         /* Write header... */
        00098
00099
                 "# $2 = altitude [km]\n"
00100
00101
                 "# $3 = longitude [deg]\n"
00102
                 "# $4 = latitude [deg]\n"
00103
                 "# $5 = pressure [hPa]\n"
00104
                 "# $6 = temperature [K] \n"
                 "# $7 = zonal wind [m/s] n"
00105
                 "# $8 = meridional wind [m/s]\n"
00106
                 "# $9 = vertical wind [hPa/s] \n'
00107
                 "# $10 = H20 volume mixing ratio [1]\n");
00109
        fprintf(out,
00110
                 "# $11 = 03 volume mixing ratio [1]\n"
                 "# $12 = geopotential height [km]\n'
00111
                 "# $13 = potential vorticity [PVU]\n"
00112
                 "# $14 = surface pressure [hPa]\n"
00113
                 "# $15 = tropopause pressure [hPa]\n"
00114
00115
                 "# $16 = tropopause geopotential height [km]\n"
00116
                 "# $17 = tropopause temperature [K]\n");
00117
00118
        /* Write data... */
        for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
00119
00120
00121
          for (ix = 0; ix < met->nx; ix++)
            00122
00123
00124
00125
00127
                        vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00128
                        h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
                       zm[ix][iy] / np[ix][iy], pvm[ix][iy] / np[ix][iy],
psm[ix][iy] / np[ix][iy], ptm[ix][iy] / np[ix][iy],
ztm[ix][iy] / np[ix][iy], ttm[ix][iy] / np[ix][iy]);
00129
00130
00131
00132
          for (ix = 0; ix < met->nx; ix++)
            00133
00134
00135
                        timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
00136
                        met->lon[ix], met->lat[iy], met->p[ip],
                        tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00137
00138
                        h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00139
                        zm[ix][iy] / np[ix][iy], pvm[ix][iy] / np[ix][iy],
psm[ix][iy] / np[ix][iy], ptm[ix][iy] / np[ix][iy],
ztm[ix][iy] / np[ix][iy], ttm[ix][iy] / np[ix][iy]);
00140
00141
00142
00143
00144
         /* Close file... */
00146
        fclose(out);
00147
        /* Free... */
00148
00149
        free (met);
00150
00151
        return EXIT_SUCCESS;
00152 }
```

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

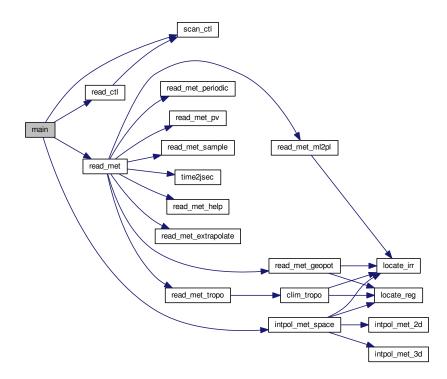
5.25.2.1 int main (int argc, char * argv[])

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],
             lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ], w, wm[NZ], h2o, h2om[NZ], o3, o3m[NZ], ps, psm[NZ], pt, ptm[NZ], tt, ttm[NZ],
00049
00050
00051
             zg, zgm[NZ], zt, ztm[NZ], pv, pvm[NZ];
00053
          static int i, iz, np[NZ];
00054
00055
           /* Allocate... */
00056
          ALLOC(met, met_t, 1);
00057
00058
           /* Check arguments... */
00059
          if (argc < 4)
00060
             ERRMSG("Give parameters: <ctl>   <met0> [ <met1> ... ]");
00061
00062
          /* Read control parameters... */
read_ctl(argv[1], argc, argv, &ctl);
z0 = scan_ctl(argv[1], argc, argv, "Z0", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "Z1", -1, "60", NULL);
dz = scan_ctl(argv[1], argc, argv, "DZ", -1, "1", NULL);
lon0 = scan_ctl(argv[1], argc, argv, "LON0", -1, "0", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "LON1", -1, "0", NULL);
dlon = scan_ctl(argv[1], argc, argv, "DLON", -1, "1", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "LAT0", -1, "0", NULL);
dlat1 = scan_ctl(argv[1], argc, argv, "DLAT", -1, "1", NULL);
dlat1 = scan_ctl(argv[1], argc, argv, "DLAT", -1, "1", NULL);
          /* Read control parameters... */
00063
00064
00066
00067
00068
00069
00070
00071
00072
           dlat = scan_ctl(argv[1], argc, argv, "DLAT", -1, "1", NULL);
00073
00074
           /* Loop over input files... */
00075
           for (i = 3; i < argc; i++) {
00076
00077
              /\star Read meteorological data... \star/
00078
             read_met(&ctl, argv[i], met);
00079
08000
              /* Average... */
              for (z = z0; z <= z1; z += dz) {
00081
00082
                iz = (int) ((z - z0) / dz);
                 if (iz < 0 || iz > NZ)
00083
00084
                   ERRMSG("Too many altitudes!");
                 for (lon = lon0; lon <= lon1; lon += dlon)
    for (lat = lat0; lat <= lat1; lat += dlat) {</pre>
00085
00086
                     00087
00088
00089
00090
                                             &tt, NULL, NULL, NULL, NULL, NULL, NULL);
00091
                      if (gsl_finite(t) && gsl_finite(u)
                            && gsl_finite(v) && gsl_finite(w)) {
00092
                        timem[iz] += met->time;
lonm[iz] += lon;
latm[iz] += lat;
00093
00094
00095
                         zgm[iz] += zg;
00096
00097
                         tm[iz] += t;
00098
                         um[iz] += u;
                         vm[iz] += v;
00099
00100
                         wm[iz] += w;
                         pvm[iz] += pv;
00101
                         h2om[iz] += h2o;
00102
                         o3m[iz] += o3;
00103
00104
                         psm[iz] += ps;
00105
                         ptm[iz] += pt;
                         ztm[iz] += zt;
ttm[iz] += tt;
00106
00107
00108
                         np[iz]++;
00109
00110
                   }
00111
             }
00112
00113
00114
          /* Create output file... */
00115
          printf("Write meteorological data file: %s\n", argv[2]);
```

```
00116
        if (!(out = fopen(argv[2], "w")))
00117
           ERRMSG("Cannot create file!");
00118
00119
         /* Write header... */
00120
        fprintf(out,
    "# $1
00121
                         = time [s]\n"
00122
                  "# $2
                         = altitude [km]\n"
00123
                  "# $3
                         = longitude [deg]\n"
00124
                  "# $4 = latitude [deg]\n"
                  "# $5 = pressure [hPa]\n"
00125
                  "# $6 = temperature [K]\n"
"# $7 = zonal wind [m/s]\n"
00126
00127
                  "# $8 = meridional wind [m/s]\n"
00128
00129
                  "# $9 = vertical wind [hPa/s]\n");
00130
        fprintf(out,
00131
                  "# $10 = H2O volume mixing ratio [1]\n"
                  "# $10 = H20 VOLUME MIXING TALLO [1]\n"
"# $11 = O3 VOLUME mixing ratio [1]\n"
"# $12 = geopotential height [km]\n"
"# $13 = potential vorticity [PVU]\n"
00132
00133
00134
00135
                  "# $14 = surface pressure [hPa]\n"
                  "# $15 = tropopause pressure [hPa]\n"
"# $16 = tropopause geopotential height [km]\n"
00136
00137
                  "# $17 = tropopause temperature [K]\n\n");
00138
00139
00140
         /* Write data... */
00141
        for (z = z0; z <= z1; z += dz) {
  iz = (int) ((z - z0) / dz);
00142
          00143
00144
00145
00146
00147
00148
00149
00150
         /* Close file... */
00151
00152
        fclose(out);
00153
00154
         /* Free... */
00155
        free (met);
00156
00157
         return EXIT_SUCCESS;
00158 }
```

Here is the call graph for this function:



5.26 met prof.c 193

5.26 met_prof.c

```
00001 /*
          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
          the Free Software Foundation, either version 3 of the License, or
00006
00007
          (at your option) any later version.
80000
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
          You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
00016
          Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00017
00018 */
00019
00025 #include "libtrac.h"
00027 /*
           Dimensions...
00028
00029
00030
00031 /* Maximum number of altitudes. */
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
          static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ], w,
wm[NZ], h2o, h2om[NZ], o3, o3m[NZ], ps, psm[NZ], pt, ptm[NZ], tt, ttm[NZ],
00048
00049
00050
00051
            zg, zgm[NZ], zt, ztm[NZ], pv, pvm[NZ];
00052
00053
          static int i, iz, np[NZ];
00054
          /* Allocate... */
00055
00056
          ALLOC(met, met_t, 1);
00057
00058
          /* Check arguments... */
00059
00060
            ERRMSG("Give parameters: <ctl>  <met0> [ <met1> ... ]");
00061
00062
          /* Read control parameters... */
00063
          read_ctl(argv[1], argc, argv, &ctl);
          read_ct1(argv[1], argc, argv, &ct1);
z0 = scan_ct1(argv[1], argc, argv, "20", -1, "0", NULL);
z1 = scan_ct1(argv[1], argc, argv, "21", -1, "60", NULL);
dz = scan_ct1(argv[1], argc, argv, "DZ", -1, "1", NULL);
lon0 = scan_ct1(argv[1], argc, argv, "LONO", -1, "0", NULL);
lon1 = scan_ct1(argv[1], argc, argv, "LONO", -1, "0", NULL);
dlon = scan_ct1(argv[1], argc, argv, "DLON", -1, "1", NULL);
00064
00065
00066
00067
00068
00069
          lat0 = scan_ctl(argv[1], argc, argv, "LAT0", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "LAT1", -1, "0", NULL);
dlat = scan_ctl(argv[1], argc, argv, "DLAT1", -1, "1", NULL);
00071
00072
00073
00074
          /* Loop over input files... */
00075
          for (i = 3; i < argc; i++) {
00076
00077
             /* Read meteorological data... */
00078
            read_met(&ctl, argv[i], met);
00079
00080
             /* Average... */
             for (z = z0; z \le z1; z += dz) {
00081
                iz = (int) ((z - z0) / dz);
00082
00083
                if (iz < 0 || iz > NZ)
00084
                  ERRMSG("Too many altitudes!");
               00085
00086
00087
00088
00089
                     intpol_met_space(met, pt, lon, lat, NULL, NULL, &zt,
```

```
&tt, NULL, NULL, NULL, NULL, NULL, NULL);
                if (gsl_finite(t) && gsl_finite(u)
    && gsl_finite(v) && gsl_finite(w)) {
00091
00092
                  timem[iz] += met->time;
lonm[iz] += lon;
latm[iz] += lat;
00093
00094
00095
                  zgm[iz] += zg;
00097
                  tm[iz] += t;
00098
                  um[iz] += u;
00099
                  vm[iz] += v;
                  wm[iz] += w;
00100
                  pvm[iz] += pv;
00101
                  h2om[iz] += h2o;
00102
00103
                  o3m[iz] += o3;
00104
                  psm[iz] += ps;
                  ptm[iz] += pt;
00105
                  ztm[iz] += zt;
00106
00107
                  ttm[iz] += tt;
                  np[iz]++;
00108
00109
                }
00110
              }
00111
          }
00112
00113
       /* Create output file... */
00114
       printf("Write meteorological data file: %s\n", argv[2]);
00115
00116
        if (!(out = fopen(argv[2], "w")))
00117
          ERRMSG("Cannot create file!");
00118
        /* Write header... */
00119
00120
       fprintf(out,
00121
                       = time [s]\n"
00122
                "# $2 = altitude [km] \n"
00123
                "# $3 = longitude [deg] \n"
                "# $4 = latitude [deg]\n"
00124
                "# $5 = pressure [hPa]\n"
00125
                "# $6 = temperature [K]\n"
00126
                "# $7 = zonal wind [m/s]\n"
00128
                "# $8
                      = meridional wind [m/s] n"
00129
                "# $9 = vertical wind [hPa/s]\n");
       fprintf(out,
    "# $10 = H2O volume mixing ratio [1]\n"
00130
00131
                "# $11 = 03 volume mixing ratio [1]\n'
00132
                "# $12 = geopotential height [km]\n'
00133
00134
                "# $13 = potential vorticity [PVU]\n"
00135
                "# $14 = surface pressure [hPa]\n"
00136
                "# $15 = tropopause pressure [hPa] \n"
                "# $16 = tropopause geopotential height [km]\n"
00137
                "# $17 = tropopause temperature [K]\n\n");
00138
00139
       /* Write data... */
for (z = z0; z <= z1; z += dz) {
00140
00141
          iz = (int) ((z - z0) / dz);
00142
         00143
00144
00145
00147
00148
00149
00150
        /* Close file... */
00151
00152
       fclose(out);
00153
       /* Free... */
00154
00155
       free(met);
00156
        return EXIT SUCCESS:
00157
00158 }
```

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

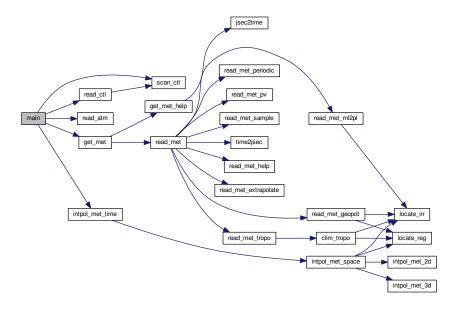
5.27.2.1 int main (int argc, char * argv[])

Definition at line 31 of file met_sample.c.

```
00033
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
00049
          ERRMSG("Give parameters: <ctl> <metbase> <atm_in> <sample.tab>");
00050
        /* Allocate... */
00051
00052
        ALLOC(atm, atm_t, 1);
        ALLOC(met0, met_t, 1);
00053
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, "MET_SAMPLE_GEOPOT", -1, "0", NULL);
00060
00061
        /* Read atmospheric data... */
00062
        read_atm(argv[3], &ctl, atm);
00063
00064
        /* Create output file... */
00065
        printf("Write meteorological data file: %s\n", argv[4]);
00066
           (!(out = fopen(argv[4], "w")))
00067
          ERRMSG("Cannot create file!");
00068
00069
        /* Write header... */
00070
        fprintf(out,
    "# $1
00071
                       = time [s]\n"
00072
                 "# $2 = altitude [km] \n"
00073
                 "# $3 = longitude [deg] \n"
                 "# $4 = latitude [deg]\n"
00074
                 "# $5 = pressure [hPa]\n"
00075
00076
                 "# $6 = temperature [K]\n"
                 "# $7 = zonal wind [m/s]\n"
00077
                       = meridional wind [m/s]\n"
00079
                "# $9 = vertical wind [hPa/s] n");
08000
        fprintf(out,
                 "# $10 = H2O volume mixing ratio [1]\n"
00081
                 "# $11 = 03 volume mixing ratio [1]\n'
00082
                "# $12 = geopotential height [km]\n"
"# $13 = potential vorticity [PVU]\n"
00083
00084
00085
                 "# $14 = surface pressure [hPa]\n"
                "# $15 = tropopause pressure [hPa]\n"
"# $16 = tropopause geopotential height [km]\n"
00086
00087
                "# $17 = tropopause temperature [K]\n\n");
00088
00089
00090
        /* Loop over air parcels... */
00091
        for (ip = 0; ip < atm->np; ip++) {
00092
00093
          /\star Get meteorological data... \star/
00094
          get_met(&ctl, argv[2], atm->time[ip], &met0, &met1);
00095
00096
          /* Set reference pressure for interpolation... */
00097
          pref = atm->p[ip];
```

```
if (geopot) {
00099
           zref = Z(pref);
00100
            p0 = met0 -> p[0];
            p1 = met0->p[met0->np - 1];
00101
            for (it = 0; it < 24; it++) {
  pref = 0.5 * (p0 + p1);
00102
00103
             intpol_met_time(met0, met1, atm->time[ip], pref, atm->
00104
00105
                             atm->lat[ip], NULL, NULL, &zm, NULL, NULL, NULL, NULL,
00106
                             NULL, NULL, NULL);
              if (zref > zm || !gsl_finite(zm))
00107
00108
             p0 = pref;
else
00109
00110
               p1 = pref;
00111
00112
           pref = 0.5 * (p0 + p1);
00113
00114
00115
          /* Interpolate meteorological data... */
         intpol_met_time(met0, met1, atm->time[ip], pref, atm->lon[ip],
00116
00117
                         atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &pv, &h2o,
00118
                         &o3);
00119
          intpol_met_time(met0, met1, atm->time[ip], pt, atm->lon[ip], atm->
     lat[ip],
00120
                         NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL, NULL);
00121
00122
          /* Write data... */
          00123
00124
                 atm->p[ip], t, u, v, w, h2o, o3, z, pv, ps, pt, zt, tt);
00125
00126
00127
00128
       /* Close file... */
00129
       fclose(out);
00130
       /* Free... */
00131
00132
       free (atm);
00133
       free (met0);
00134
       free (met1);
00135
00136
       return EXIT_SUCCESS;
00137 }
```

Here is the call graph for this function:



5.28 met_sample.c

00001 /*

5.28 met sample.c 197

```
00002
        This file is part of MPTRAC.
00003
00004
        MPTRAC is free software: you can redistribute it and/or modify
        it under the terms of the GNU General Public License as published by
00005
        the Free Software Foundation, either version 3 of the License, or
00006
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
        You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
00016
00017
        Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /*
         Main...
00028
00029
00030
00031 int main(
00032
        int argc,
        char *argv[]) {
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> <metbase> <atm in> <sample.tab>");
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
00059
          (int) scan_ctl(argv[1], argc, argv, "MET_SAMPLE_GEOPOT", -1, "0", NULL);
00060
00061
        /* Read atmospheric data... */
00062
        read_atm(argv[3], &ctl, atm);
00063
00064
        /* Create output file... */
00065
        printf("Write meteorological data file: s\n", argv[4]);
00066
        if (!(out = fopen(argv[4], "w")))
          ERRMSG("Cannot create file!");
00067
00068
00069
        /* Write header... */
00070
        fprintf(out,
00071
                 "# $1
                        = time [s]\n"
                 "# $2 = altitude [km]\n"
00072
                 "# $3 = longitude [deg]\n"
"# $4 = latitude [deg]\n"
00073
00074
                 "# $5 = pressure [hPa]\n"
00075
                 "# $6 = temperature [K]\n"
00076
00077
                 "# $7 = zonal wind [m/s] \n"
                 "# $8 = meridional wind [m/s]\n"
00078
                "# $9 = vertical wind [hPa/s]\n");
00079
00080
        fprintf(out,
                 "# $10 = H20 volume mixing ratio [1]\n"
00081
                 "# $11 = 03 volume mixing ratio [1]\n
00082
00083
                 "# $12 = geopotential height [km] \n"
00084
                 "# $13 = potential vorticity [PVU]\n"
                 "# $14 = surface pressure [hPa]\n"
00085
                 "# $15 = tropopause pressure [hPa]\n"
00086
                 "# $16 = tropopause geopotential height [km]\n"
00087
00088
                 "# $17 = tropopause temperature [K]\n\n");
00089
        /* Loop over air parcels... */
00090
00091
        for (ip = 0; ip < atm->np; ip++) {
00092
00093
          /* Get meteorological data... */
```

```
get_met(&ctl, argv[2], atm->time[ip], &met0, &met1);
00095
00096
          /\star Set reference pressure for interpolation... \star/
00097
          pref = atm->p[ip];
00098
          if (geopot) {
  zref = Z(pref);
00099
00100
           p0 = met0 - > p[0];
00101
            p1 = met0 - p[met0 - pnp - 1];
            for (it = 0; it < 24; it++)
  pref = 0.5 * (p0 + p1);</pre>
00102
00103
              intpol_met_time(met0, met1, atm->time[ip], pref, atm->
00104
     lon[ip],
                              atm->lat[ip], NULL, NULL, &zm, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
00105
00106
00107
              if (zref > zm || !gsl_finite(zm))
             p0 = pref;
else
00108
00109
           p1 = pref;
00110
00111
00112
           pref = 0.5 * (p0 + p1);
00113
00114
00115
          /\star Interpolate meteorological data... \star/
00116
         intpol_met_time(met0, met1, atm->time[ip], pref, atm->lon[ip],
00117
                          atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &pv, &h2o,
00118
                           &o3);
00119
         intpol_met_time(met0, met1, atm->time[ip], pt, atm->lon[ip], atm->
     lat[ip],
00120
                          NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL, NULL);
00121
         00122
00123
00124
                 atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
00125
                  atm->p[ip], t, u, v, w, h2o, o3, z, pv, ps, pt, zt, tt);
00126
00127
00128
       /* Close file... */
00129
       fclose(out);
00130
00131
        /* Free... */
00132
       free (atm);
00133
       free (met0);
00134
       free (met1);
00135
00136
       return EXIT_SUCCESS;
00137 }
```

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

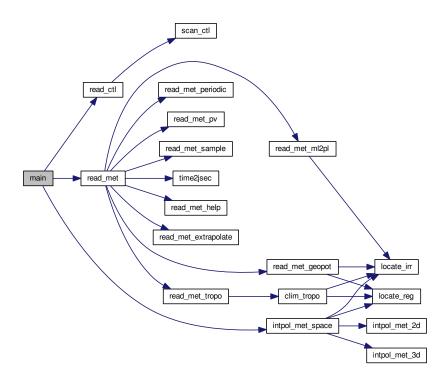
5.29.2.1 int main (int argc, char * argv[])

Definition at line 27 of file met zm.c.

```
00029
00030
00031
        ctl_t ctl;
00032
00033
        met t *met;
00034
        FILE *out;
00036
00037
        static double timem[EP][EY], psm[EP][EY], ptm[EP][EY], ttm[EP][EY],
          ztm[EP][EY], tm[EP][EY], um[EP][EY], vm[EP][EY], wm[EP][EY], h2om[EP][EY], pvm[EP][EY], o3m[EP][EY], zm[EP][EY], zt, tt;
00038
00039
00040
00041
        static int i, ip, ix, iy, np[EP][EY], npt[EP][EY];
00042
00043
         /* Allocate... */
00044
        ALLOC(met, met_t, 1);
00045
00046
        /* Check arguments... */
00047
            (argc < 4)
00048
          ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00049
00050
        /\star Read control parameters... \star/
00051
        read_ctl(argv[1], argc, argv, &ctl);
00052
00053
         /* Loop over files... */
00054
        for (i = 3; i < argc; i++) {
00055
00056
           /\star Read meteorological data... \star/
00057
          read_met(&ctl, argv[i], met);
00058
00059
           /* Average data... */
           for (ix = 0; ix < met->nx; ix++)
00060
00061
             for (iy = 0; iy < met->ny; iy++)
00062
               for (ip = 0; ip < met->np; ip++) {
00063
                 intpol_met_space(met, met->pt[ix][iy], met->lon[ix], met->
      lat[iy],
00064
                                    NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL,
00065
                                    NULL);
00066
                 timem[ip][iy] += met->time;
                 zm[ip][iy] += met->z[ix][iy][ip];
tm[ip][iy] += met->t[ix][iy][ip];
00067
00068
00069
                 um[ip][iy] += met->u[ix][iy][ip];
                 vm[ip][iy] += met->v[ix][iy][ip];
00070
                 wm[ip][iy] += met->w[ix][iy][ip];
00071
00072
                 pvm[ip][iy] += met->pv[ix][iy][ip];
00073
                 h2om[ip][iy] += met->h2o[ix][iy][ip];
                 o3m[ip][iy] += met->o3[ix][iy][ip];
psm[ip][iy] += met->ps[ix][iy];
00074
00075
00076
                 if (gsl_finite(met->pt[ix][iy]))
00077
                  ptm[ip][iy] += met->pt[ix][iy];
00078
                   ztm[ip][iy] += zt;
00079
                   ttm[ip][iy] += tt;
00080
                   npt[ip][iy]++;
00081
00082
                 np[ip][iy]++;
00083
00084
        }
00085
00086
        /* Create output file... */
        printf("Write meteorological data file: %s\n", argv[2]);
if (!(out = fopen(argv[2], "w")))
00087
00088
00089
          ERRMSG("Cannot create file!");
00090
00091
         /* Write header... */
00092
        fprintf(out,
00093
                  "# $1
                        = time [s]\n"
                 "# $2 = altitude [km]\n"
00094
                  "# $3 = longitude [deg] \n"
00095
                 "# $4 = latitude [deg]\n"
00096
00097
                 "# $5 = pressure [hPa]\n"
00098
                 "# $6
                        = temperature [K]\n"
                 "# $7 = zonal wind [m/s]\n"
00099
                  "# $8 = meridional wind [m/s]\n"
00100
00101
                 "# $9 = vertical wind [hPa/s]\n'
00102
                 "# $10 = H2O volume mixing ratio [1]\n");
00103
        fprintf(out,
```

```
"# $11 = 03 volume mixing ratio [1]\n"
                   "# $12 = geopotential height [km]\n"
"# $13 = potential vorticity [PVU]\n"
00105
00106
                    "# $14 = surface pressure [hPa]\n"
00107
                    "# $15 = tropopause pressure [hPa]\n"
00108
                    "# $15 = tropopause geopotential height [km]\n"
"# $17 = tropopause temperature [K]\n");
00109
00110
00111
         00112
00113
00114
00115
00116
00117
00118
                         met->p[ip], tm[ip][iy] / np[ip][iy], um[ip][iy] / np[ip][iy],
                         vm[ip][iy] / np[ip][iy], wm[ip][iy] / np[ip][iy],
00119
                        vm(ip)[iy] / inp[ip][iy], vm(ip)[iy] / np[ip][iy],
2m[ip][iy] / np[ip][iy], pvm[ip][iy] / np[ip][iy],
psm[ip][iy] / np[ip][iy], ptm[ip][iy] / npt[ip][iy],
ztm[ip][iy] / npt[ip][iy], ttm[ip][iy] / npt[ip][iy]);
00120
00121
00122
00123
00124
00125
         /* Close file... */
00126
00127
         fclose(out);
00128
00129
          /* Free... */
00130
         free (met);
00131
00132
         return EXIT_SUCCESS;
00133 }
```

Here is the call graph for this function:



5.30 met_zm.c

```
00001 /\star
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
```

5.30 met zm.c 201

```
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,
        but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
00012
        GNU General Public License for more details.
00013
00014
        You should have received a copy of the GNU General Public License
        along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>.
00015
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
        met_t *met;
00034
00035
        FILE *out;
00036
00037
        static double timem[EP][EY], psm[EP][EY], ptm[EP][EY], ttm[EP][EY],
00038
         ztm[EP][EY], tm[EP][EY], um[EP][EY], vm[EP][EY], wm[EP][EY], h2om[EP][EY],
00039
          pvm[EP][EY], o3m[EP][EY], zm[EP][EY], zt, tt;
00040
00041
        static int i, ip, ix, iy, np[EP][EY], npt[EP][EY];
00042
00043
         /* Allocate... */
00044
        ALLOC(met, met_t, 1);
00045
00046
         /* Check arguments... */
00047
        if (argc < 4)
00048
           ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00049
00050
         /* Read control parameters... */
00051
        read_ctl(argv[1], argc, argv, &ctl);
00052
        /* Loop over files... */
for (i = 3; i < argc; i++) {</pre>
00053
00054
00055
00056
           /* Read meteorological data... */
00057
           read_met(&ctl, argv[i], met);
00058
00059
           /* Average data... */
           for (ix = 0; ix < met->nx; ix++)
00060
             for (iy = 0; iy < met->ny; iy++)
00061
00062
               for (ip = 0; ip < met->np; ip++) {
00063
                 intpol_met_space(met, met->pt[ix][iy], met->lon[ix], met->
      lat[iy],
00064
                                     NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL,
00065
                                     NULL);
00066
                  timem[ip][iy] += met->time;
                  zm[ip][iy] += met ->z[ix][iy][ip];
tm[ip][iy] += met ->t[ix][iy][ip];
00067
00068
                  um[ip][iy] += met->u[ix][iy][ip];
vm[ip][iy] += met->v[ix][iy][ip];
00069
00070
                  wm[ip][iy] += met->w[ix][iy][ip];
00071
00072
                  pvm[ip][iy] += met->pv[ix][iy][ip];
00073
                  h2om[ip][iy] += met->h2o[ix][iy][ip];
00074
                  o3m[ip][iy] += met->o3[ix][iy][ip];
                  psm[ip][iy] += met->ps[ix][iy];
00075
00076
                  if (gsl_finite(met->pt[ix][iy]))
                   ptm[ip][iy] += met->pt[ix][iy];
ztm[ip][iy] += zt;
00077
00078
                    ttm[ip][iy] += tt;
00079
08000
                    npt[ip][iy]++;
00081
00082
                  np[ip][iy]++;
00083
00084
        }
00085
00086
         /* Create output file... */
00087
        printf("Write meteorological data file: s\n", argv[2]);
00088
         if (!(out = fopen(argv[2], "w")))
          ERRMSG("Cannot create file!");
00089
00090
00091
         /* Write header... */
00092
        fprintf(out,
00093
                  "# $1
                         = time [s]\n"
                  "# $2 = altitude [km]\n"
00094
                  "# $3 = longitude [deg]\n"
"# $4 = latitude [deg]\n"
00095
00096
```

```
"# $5 = pressure [hPa]\n'
00098
                    "# $6 = temperature [K] \n"
                    "# $7 = zonal wind [m/s]\n"
00099
                    "# $8 = meridional wind [m/s]\n"
00100
                    "# $9 = vertical wind [hPa/s]\n'
00101
                    "# $10 = H2O volume mixing ratio [1]\n");
00102
00103 fprintf(out,
00104 "# $11 = 03 volume mixing ratio [1]\n"
                    "# $12 = geopotential height [km]\n"
00105
                    "# $13 = potential vorticity [PVU]\n"
00106
                    "# $14 = surface pressure [hPa]\n"
00107
                    "# $15 = tropopause pressure [hPa]\n"
00108
                    "# $16 = tropopause geopotential height [km]\n"
"# $17 = tropopause temperature [K]\n");
00109
00110
00111
         00112
00113
00114
00115
00116
00117
                         met->p[ip], tm[ip][iy] / np[ip][iy], um[ip][iy] / np[ip][iy],
vm[ip][iy] / np[ip][iy], wm[ip][iy] / np[ip][iy],
00118
00119
                        tant[p][iy] / np[ip][iy], man[ip][iy] / np[ip][iy],
th2om[ip][iy] / np[ip][iy], o3m[ip][iy] / np[ip][iy],
zm[ip][iy] / np[ip][iy], pvm[ip][iy] / np[ip][iy],
psm[ip][iy] / np[ip][iy], ptm[ip][iy] / npt[ip][iy],
ztm[ip][iy] / npt[ip][iy], ttm[ip][iy] / npt[ip][iy]);
00120
00121
00122
00123
00124
00125
         /* Close file... */
00126
00127
         fclose(out);
00128
00129
         /* Free... */
00130
         free (met);
00131
00132
          return EXIT_SUCCESS;
00133 }
```

5.31 smago.c File Reference

Estimate horizontal diffusivity based on Smagorinsky theory.

Functions

• int main (int argc, char *argv[])

5.31.1 Detailed Description

Estimate horizontal diffusivity based on Smagorinsky theory.

Definition in file smago.c.

5.31.2 Function Documentation

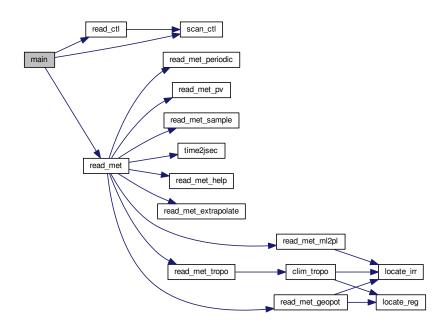
5.31.2.1 int main (int argc, char * argv[])

Definition at line 27 of file smago.c.

```
00029
00030
00031
         ctl_t ctl;
00032
00033
         met t *met;
00034
        FILE *out;
00036
00037
         static double dz, dzmin = 1e10, z, t, s, 1s2, k[EX][EY], c = 0.15;
00038
00039
         static int ip, ip2, ix, iy;
00040
00041
         /* Allocate... */
00042
         ALLOC(met, met_t, 1);
00043
00044
         /* Check arguments... */
00045
         if (argc < 4)
           ERRMSG("Give parameters: <ctl> <map.tab> <met>");
00046
00047
00048
         /* Read control parameters... */
00049
        read_ctl(argv[1], argc, argv, &ctl);
        z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00050
00051
00052
         /* Read meteorological data... */
00053
         read_met(&ctl, argv[3], met);
00054
00055
         /\star Find nearest pressure level... \star/
00056
         for (ip2 = 0; ip2 < met->np; ip2++) {
          dz = fabs(Z(met->p[ip2]) - z);
00057
00058
           if (dz < dzmin) {</pre>
             dzmin = dz;
00059
00060
             ip = ip2;
00061
00062
00063
00064
         /* Write info... */
00065
         printf("Analyze %g hPa...\n", met->p[ip]);
00066
00067
         /* Calculate horizontal diffusion coefficients... */
         for (ix = 1; ix < met->nx - 1; ix++)
for (iy = 1; iy < met->ny - 1; iy++) {
00068
00069
             t = 0.5 * ((met > u[ix + 1][iy][ip] - met > u[ix - 1][iy][ip])
00070
00071
                         / (1000. *
00072
                             DEG2DX(met->lon[ix + 1] - met->lon[ix - 1], met->lat[iy]))
00073
                          - (met->v[ix][iy + 1][ip] - met->v[ix][iy - 1][ip])
00074
                          / (1000. * DEG2DY(met->lat[iy + 1] - met->lat[iy
                                                                                    11)));
00075
              s = 0.5 * ((met->u[ix][iy + 1][ip] - met->u[ix][iy - 1][ip])
                          / (1000. * DEG2DY(met->lat[iy + 1] - met->lat[iy - 1]))
00076
                          + (met->v[ix + 1][iy][ip] - met->v[ix - 1][iy][ip])
00077
00078
                          / (1000. *
00079
                             DEG2DX(met->lon[ix + 1] - met->lon[ix - 1],
00080
                                    met->lat[iy])));
00081
              ls2 = SQR(c * 500. * DEG2DY(met->lat[iy + 1] - met->lat[iy - 1]));
             if (fabs(met->lat[iy]) > 80)
  ls2 *= (90. - fabs(met->lat[iy])) / 10.;
k[ix][iy] = ls2 * sqrt(2.0 * (SQR(t) + SQR(s)));
00082
00083
00084
00085
00086
00087
         /* Create output file... */
         printf("Write data file: %s\n", argv[2]);
if (!(out = fopen(argv[2], "w")))
00088
00089
           ERRMSG("Cannot create file!");
00090
00091
00092
         /* Write header... */
00093
         fprintf(out,
00094
                  "# $1 = longitude [deg] \n"
                  "# $2 = latitude [deg] \n"
00095
                  "# $3 = zonal wind [m/s]\n"
"# $4 = meridional wind [m/s]\n"
00096
00097
                  "# $5 = \text{horizontal diffusivity } [m^2/s] \n");
00098
00099
00100
         /* Write data... */
         for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
  for (ix = 0; ix < met->nx; ix++)
00101
00102
00103
             if (met \rightarrow lon[ix] >= 180)
00104
00105
                fprintf(out, "%g %g %g %g %g\n",
00106
                        met->lon[ix] - 360.0, met->lat[iy],
                        met->u[ix][iy][ip], met->v[ix][iy][ip], k[ix][iy]);
00107
           for (ix = 0; ix < met->nx; ix++)
00108
             if (met->lon[ix] <= 180)
fprintf(out, "%g %g %g %g %g\n",
00109
00110
00111
                         met->lon[ix], met->lat[iy],
00112
                         met->u[ix][iy][ip], met->v[ix][iy][ip], k[ix][iy]);
00113
00114
         /* Close file... */
00115
```

```
00116     fclose(out);
00117
00118     /* Free... */
00119     free(met);
00120
00121     return EXIT_SUCCESS;
00122 }
```

Here is the call graph for this function:



5.32 smago.c

```
00001 /*
00002
          This file is part of MPTRAC.
00003
00004
          MPTRAC is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by
00005
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,
          but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
00012
          GNU General Public License for more details.
00013
          You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
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
          met_t *met;
00034
00035
          FILE *out:
00036
00037
          static double dz, dzmin = 1e10, z, t, s, ls2, k[EX][EY], c = 0.15;
```

5.32 smago.c 205

```
00038
00039
        static int ip, ip2, ix, iy;
00040
00041
        /* Allocate... */
00042
        ALLOC(met, met_t, 1);
00043
00044
        /* Check arguments... */
00045
        if (argc < 4)
00046
          ERRMSG("Give parameters: <ctl> <map.tab> <met>");
00047
00048
        /* Read control parameters... */
00049
        read_ctl(argv[1], argc, argv, &ctl);
00050
        z = scan_{ctl}(argv[1], argc, argv, "Z", -1, "", NULL);
00051
00052
        /* Read meteorological data... */
00053
        read_met(&ctl, argv[3], met);
00054
00055
        /* Find nearest pressure level... */
        for (ip2 = 0; ip2 < met->np; ip2++) {
00056
          dz = fabs(Z(met->p[ip2]) - z);
00057
00058
          if (dz < dzmin) {
00059
            dzmin = dz;
00060
            ip = ip2;
00061
00062
        }
00063
00064
        /* Write info... */
00065
        printf("Analyze %g hPa...\n", met->p[ip]);
00066
00067
        /* Calculate horizontal diffusion coefficients... */
        for (ix = 1; ix < met->nx - 1; ix++)
for (iy = 1; iy < met->ny - 1; iy++) {
00068
00069
00070
            t = 0.5 * ((met - u[ix + 1][iy][ip] - met - u[ix - 1][iy][ip])
00071
                        / (1000. *
                        DEG2DX(met->lon[ix + 1] - met->lon[ix - 1], met->lat[iy]))
- (met->v[ix][iy + 1][ip] - met->v[ix][iy - 1][ip])
00072
00073
00074
                        / (1000. * DEG2DY(met->lat[iy + 1] - met->lat[iy - 1])));
            s = 0.5 * ((met->u[ix][iy + 1][ip] - met->u[ix][iy - 1][ip])
00076
                        / (1000. * DEG2DY(met->lat[iy + 1] - met->lat[iy
00077
                        + (met->v[ix + 1][iy][ip] - met->v[ix - 1][iy][ip])
00078
                        / (1000. *
00079
                           DEG2DX(met->lon[ix + 1] - met->lon[ix - 1],
00080
                                  met->lat[iy])));
            ls2 = SQR(c * 500. * DEG2DY(met->lat[iy + 1] - met->lat[iy - 1]));
00081
            if (fabs(met->lat[iy]) > 80)
00082
00083
              1s2 *= (90. - fabs(met->lat[iy])) / 10.;
00084
            k[ix][iy] = 1s2 * sqrt(2.0 * (SQR(t) + SQR(s)));
00085
00086
        /* Create output file... */
printf("Write data file: %s\n", argv[2]);
00087
00088
00089
        if (!(out = fopen(argv[2], "w")))
00090
          ERRMSG("Cannot create file!");
00091
00092
        /* Write header... */
00093
        fprintf(out,
                 "# $1 = longitude [deg]\n"
00094
00095
                "# $2 = latitude [deg] \n"
00096
                "# $3 = zonal wind [m/s] \n"
                 "# $4 = meridional wind [m/s] \n"
00097
                "# $5 = \text{horizontal diffusivity } [m^2/s] \n");
00098
00099
00100
        /* Write data... */
        for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
00101
00102
          for (ix = 0; ix < met->nx; ix++)
00103
            00104
00105
00106
                      met->u[ix][iy][ip], met->v[ix][iy][ip], k[ix][iy]);
00107
00108
          for (ix = 0; ix < met->nx; ix++)
            00109
00110
00111
                      met->u[ix][iy][ip], met->v[ix][iy][ip], k[ix][iy]);
00112
00113
00114
00115
        /* Close file... */
00116
        fclose(out);
00117
        /* Free... */
00118
00119
        free (met);
00120
00121
        return EXIT_SUCCESS;
00122 }
```

5.33 time2jsec.c File Reference

Convert date to Julian seconds.

Functions

• int main (int argc, char *argv[])

5.33.1 Detailed Description

Convert date to Julian seconds.

Definition in file time2jsec.c.

5.33.2 Function Documentation

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

Definition at line 27 of file time2jsec.c.

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

Here is the call graph for this function:



5.34 time2jsec.c 207

5.34 time2jsec.c

```
00001 /*
        This file is part of MPTRAC.
00002
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.
80000
00009
        MPTRAC is distributed in the hope that it will be useful,
        but WITHOUT ANY WARRANTY; without even the implied warranty of
00010
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 <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>.
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;
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]);
        mon = atoi(argv[2]);
00041
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
       time2jsec(year, mon, day, hour, min, sec, remain, &jsec);
printf("%.2f\n", jsec);
00049
00050
00051
00052
        return EXIT SUCCESS;
```

5.35 trac.c File Reference

Lagrangian particle dispersion model.

Functions

- void module_advection (met_t *met0, met_t *met1, atm_t *atm, int ip, double dt)
 - Calculate advection of air parcels.
- void module_decay (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, int ip, double dt)

Calculate exponential decay of particle mass.

- void module_diffusion_meso (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, int ip, double dt, gsl_rng *rng)
- void module_diffusion_turb (ctl_t *ctl, atm_t *atm, int ip, double dt, gsl_rng *rng)

Calculate turbulent diffusion.

Calculate mesoscale diffusion.

void module_isosurf (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, int ip)

Force air parcels to stay on isosurface.

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

Interpolate meteorological data for air parcel positions.

- void module_position (met_t *met0, met_t *met1, atm_t *atm, int ip)

 Check position of air parcels.
- void module_sedi (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, int ip, 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[])

5.35.1 Detailed Description

Lagrangian particle dispersion model.

Definition in file trac.c.

5.35.2 Function Documentation

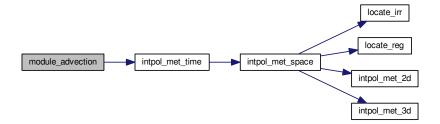
5.35.2.1 void module_advection ($met_t * met0$, $met_t * met1$, $atm_t * atm$, int ip, double dt)

Calculate advection of air parcels.

Definition at line 386 of file trac.c.

```
00392
00393
        double v[3], xm[3];
00394
00395
         /* Interpolate meteorological data... */
        00396
00397
00398
00399
        /\star Get position of the mid point... \star/
00400
        xm[0] = atm->lon[ip] + DX2DEG(0.5 * dt * v[0] / 1000., atm->lat[ip]);
xm[1] = atm->lat[ip] + DY2DEG(0.5 * dt * v[1] / 1000.);
xm[2] = atm->p[ip] + 0.5 * dt * v[2];
00401
00402
00403
00404
00405
         /* Interpolate meteorological data for mid point...
        00406
00407
                          &v[0], &v[1], &v[2], NULL, NULL, NULL);
00408
00409
00410
        /* Save new position... */
00411
        atm->time[ip] += dt;
        atm->lon[ip] += DX2DEG(dt * v[0] / 1000., xm[1]);
atm->lat[ip] += DY2DEG(dt * v[1] / 1000.);
atm->p[ip] += dt * v[2];
00412
00413
00414
00415 }
```

Here is the call graph for this function:



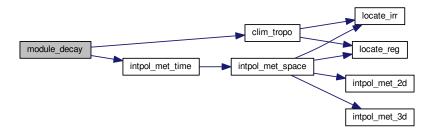
5.35.2.2 void module_decay ($ctl_t * ctl$, $met_t * met0$, $met_t * met1$, $atm_t * atm$, it ip, double dt)

Calculate exponential decay of particle mass.

Definition at line 419 of file trac.c.

```
00425
                     {
00426
00427
        double ps, pt, tdec;
00428
00429
         /* Set constant lifetime... */
00430
        if (ctl->tdec_trop == ctl->tdec_strat)
00431
          tdec = ctl->tdec_trop;
00432
00433
        /* Set altitude-dependent lifetime... */
00434
        else {
00435
00436
            /* Get surface pressure... */
00437
           intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
                             atm->lon[ip], atm->lat[ip], &ps, NULL, NULL);
00438
00439
00440
00441
           /* Get tropopause pressure... */
00442
           pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00443
           /* Set lifetime... */
if (atm->p[ip] <= pt)</pre>
00444
00445
00446
             tdec = ctl->tdec_strat;
00448
             tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
00449
00450
00451
         /* Calculate exponential decay... */
         atm \rightarrow q[ctl \rightarrow qnt_m][ip] *= exp(-dt / tdec);
00453 }
```

Here is the call graph for this function:



5.35.2.3 void module_diffusion_meso (ctl_t * ctl, met_t * met0, met_t * met1, atm_t * atm, int ip, double dt, gsl_rng * rng)

Calculate mesoscale diffusion.

Definition at line 457 of file trac.c.

```
00464 {
00465
00466 double r, rs, u[16], v[16], w[16];
00467
00468 int ix, iy, iz;
00469
00470 /* Get indices... */
```

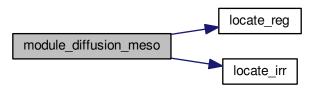
```
ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
00472
         iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
00473
         iz = locate_irr(met0->p, met0->np, atm->p[ip]);
00474
00475
         /* Caching of wind standard deviations... */
00476
         if (atm->cache_time[ix][iy][iz] != met0->time) {
00477
00478
            /* Collect local wind data... */
00479
           u[0] = met0 -> u[ix][iy][iz];
           u[1] = met0 -> u[ix + 1][iy][iz];
00480
           u[3] = met0->u[ix][iy + 1][iz];

u[3] = met0->u[ix + 1][iy + 1][iz];

u[4] = met0->u[ix][iy][iz + 1];
00481
00482
00483
00484
           u[5] = met0 -> u[ix + 1][iy][iz + 1];
00485
           u[6] = met0 -> u[ix][iy + 1][iz + 1];
           u[7] = met0 -> u[ix + 1][iy + 1][iz + 1];
00486
00487
00488
           v[0] = met0 -> v[ix][iy][iz];
           v[1] = met0 -> v[ix + 1][iy][iz];
00490
           v[2] = met0 -> v[ix][iy + 1][iz];
           v[3] = met0->v[ix + 1][iy + 1][iz];
v[4] = met0->v[ix][iy][iz + 1];
00491
00492
           v[5] = met0 -> v[ix + 1][iy][iz + 1];
00493
           v[6] = met0->v[ix][iy + 1][iz + 1];
v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00494
00495
00496
           w[0] = met0->w[ix][iy][iz];
00497
00498
           w[1] = met0->w[ix + 1][iy][iz];
00499
           w[2] = met0 -> w[ix][iy + 1][iz];
           w[3] = met0->w[ix + 1][iy + 1][iz];
w[4] = met0->w[ix][iy][iz + 1];
00500
00501
00502
           w[5] = met0 -> w[ix + 1][iy][iz + 1];
00503
           w[6] = met0 -> w[ix][iy + 1][iz + 1];
00504
           w[7] = met0 -> w[ix + 1][iy + 1][iz + 1];
00505
00506
           /* Collect local wind data... */
           u[8] = met1->u[ix][iy][iz];
u[9] = met1->u[ix + 1][iy][iz];
00507
00509
           u[10] = met1->u[ix][iy + 1][iz];
           u[11] = met1->u[ix + 1][iy + 1][iz];
u[12] = met1->u[ix][iy][iz + 1];
00510
00511
           u[13] = met1->u[ix + 1][iy][iz + 1];
u[14] = met1->u[ix][iy + 1][iz + 1];
00512
00513
00514
           u[15] = met1 -> u[ix + 1][iy + 1][iz + 1];
00515
           v[8] = met1->v[ix][iy][iz];
00516
00517
           v[9] = met1->v[ix + 1][iy][iz];
00518
           v[10] = met1->v[ix][iy + 1][iz];
           v[11] = met1->v[ix + 1][iy + 1][iz];
v[12] = met1->v[ix][iy][iz + 1];
00519
00520
           v[13] = met1 -> v[ix + 1][iy][iz + 1];
00522
           v[14] = met1->v[ix][iy + 1][iz + 1];
00523
           v[15] = met1 -> v[ix + 1][iy + 1][iz + 1];
00524
00525
           w[8] = met1->w[ix][iy][iz];
00526
           w[9] = met1->w[ix + 1][iy][iz];
           w[10] = met1->w[ix][iy + 1][iz];
00528
           w[11] = met1->w[ix + 1][iy + 1][iz];
           w[12] = met1->w[ix][iy][iz + 1];
00529
00530
           w[13] = met1->w[ix + 1][iy][iz + 1];
           w[14] = met1->w[ix][iy + 1][iz + 1];
w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00531
00532
00534
            /* Get standard deviations of local wind data... */
00535
           atm->cache_usig[ix][iy][iz] = (float) gsl_stats_sd(u, 1, 16);
           atm->cache_vsig[ix][iy][iz] = (float) gsl_stats_sd(v, 1, 16);
00536
           atm->cache_wsig[ix][iy][iz] = (float) gsl_stats_sd(w, 1, 16);
00537
           atm->cache_time[ix][iy][iz] = met0->time;
00538
00539
00540
00541
         /\star Set temporal correlations for mesoscale fluctuations... \star/
00542
         r = 1 - 2 * fabs(dt) / ctl->dt_met;
00543
         rs = sqrt(1 - r * r);
00544
00545
         /* Calculate horizontal mesoscale wind fluctuations... */
00546
         if (ctl->turb_mesox > 0) {
00547
           atm->up[ip] = (float)
00548
              (r * atm->up[ip]
00549
               + rs * gsl_ran_gaussian_ziggurat(rng,
00550
                                                     ctl->turb mesox *
                                                     atm->cache_usig[ix][iy][iz]));
00551
           atm->lon[ip] += DX2DEG(atm->up[ip] * dt / 1000., atm->lat[ip]);
00553
00554
           atm->vp[ip] = (float)
00555
             (r * atm->vp[ip]
00556
               + rs * gsl_ran_gaussian_ziggurat(rng,
00557
                                                     ctl->turb_mesox *
```

```
atm->cache_vsig[ix][iy][iz]));
00559
          atm->lat[ip] += DY2DEG(atm->vp[ip] * dt / 1000.);
00560
00561
00562
        /* Calculate vertical mesoscale wind fluctuations... */
00563
        if (ctl->turb mesoz > 0) {
         atm->wp[ip] = (float)
00564
00565
            (r * atm->wp[ip]
00566
             + rs * gsl_ran_gaussian_ziggurat(rng,
00567
                                                 ctl->turb mesoz *
00568
                                                 atm->cache_wsig[ix][iy][iz]));
00569
          atm \rightarrow p[ip] += atm \rightarrow wp[ip] * dt;
00570
00571 }
```

Here is the call graph for this function:



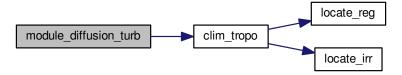
5.35.2.4 void module_diffusion_turb ($ctl_t * ctl$, $atm_t * atm$, int ip, double dt, $gsl_rng * rng$)

Calculate turbulent diffusion.

Definition at line 575 of file trac.c.

```
00580
00581
00582
        double dx, dz, pt, p0, p1, w;
00583
00584
        /* Get tropopause pressure... */
00585
        pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00586
00587
        /* Get weighting factor... */
        p1 = pt * 0.866877899;
p0 = pt / 0.866877899;
00588
00589
00590
        if (atm->p[ip] > p0)
00591
          w = 1;
00592
        else if (atm->p[ip] < p1)</pre>
00593
          w = 0;
00594
        else
00595
          w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00596
00597
         /* Set diffusivity... */
        dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00598
00599
00600
00601
        /* Horizontal turbulent diffusion... */
00602
        if (dx > 0) {
          atm->lon[ip]
00603
00604
            += DX2DEG(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00605
                        / 1000., atm->lat[ip]);
          atm->lat[ip]
00606
00607
             += DY2DEG(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00608
                        / 1000.);
00609
00610
00611
        /* Vertical turbulent diffusion... */
00612
        if (dz > 0)
          atm->p[ip]
00613
00614
             += DZ2DP(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dz * fabs(dt)))
00615
                       / 1000., atm->p[ip]);
00616 }
```

Here is the call graph for this function:



```
5.35.2.5 void module_isosurf ( ctl_t * ctl, met_t * met0, met_t * met1, atm_t * atm, int ip )
```

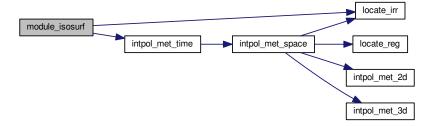
Force air parcels to stay on isosurface.

Definition at line 620 of file trac.c.

```
00625
00626
        static double *iso, *ps, t, *ts;
00628
00629
       static int idx, ip2, n;
00630
00631
       FILE *in:
00632
00633
        char line[LEN];
00634
00635
        /* Initialize... */
00636
        if (ip < 0) {</pre>
00637
00638
          /* Allocate... */
00639
          ALLOC(iso, double,
00640
                NP);
00641
          ALLOC(ps, double,
00642
                NP);
00643
          ALLOC(ts, double,
00644
                NP);
00645
00646
          /* Save pressure... */
00647
          if (ctl->isosurf == 1)
00648
            for (ip2 = 0; ip2 < atm->np; ip2++)
00649
              iso[ip2] = atm->p[ip2];
00650
00651
          /* Save density... */
00652
          else if (ctl->isosurf == 2)
00653
            for (ip2 = 0; ip2 < atm->np; ip2++) {
00654
              intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00655
              &t, NULL, NULL, NULL, NULL, NULL, NULL, NULL); iso[ip2] = atm->p[ip2] / t;
                              atm->lon[ip2], atm->lat[ip2], NULL, NULL, NULL,
00656
00657
00658
00659
00660
          /\star Save potential temperature... \star/
          else if (ctl->isosurf == 3)
    for (ip2 = 0; ip2 < atm->np; ip2++) {
00661
00662
              00663
00664
00665
00666
              iso[ip2] = THETA(atm->p[ip2], t);
00667
00668
          /\star Read balloon pressure data... \star/
00669
00670
          else if (ctl->isosurf == 4) {
00671
00672
            /* Write info... */
            printf("Read balloon pressure data: %s\n", ctl->balloon);
00673
00674
00675
            /* Open file... */
00676
            if (!(in = fopen(ctl->balloon, "r")))
00677
              ERRMSG("Cannot open file!");
```

```
00679
              /* Read pressure time series... */
             while (fgets(line, LEN, in))
  if (sscanf(line, "%lg %lg", &ts[n], &ps[n]) == 2)
00680
00681
00682
                  if ((++n) > NP)
                    ERRMSG("Too many data points!");
00683
00684
00685
              /\star Check number of points... \star/
00686
                ERRMSG("Could not read any data!");
00687
00688
              /* Close file... */
00689
00690
             fclose(in);
00691
00692
00693
           /* Leave initialization... */
           return;
00694
00695
00696
00697
         /* Restore pressure... */
00698
        if (ctl->isosurf == 1)
00699
           atm->p[ip] = iso[ip];
00700
00701
        /* Restore density... */
else if (ctl->isosurf == 2) {
00702
00703
           intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
       lon[ip],
00704
                              atm->lat[ip], NULL, NULL, NULL, &t,
00705
                              NULL, NULL, NULL, NULL, NULL, NULL);
00706
          atm \rightarrow p[ip] = iso[ip] * t;
00707
00708
00709
         /\star Restore potential temperature... \star/
00710
         else if (ctl->isosurf == 3) {
00711
           intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
       lon[ip],
                              atm->lat[ip], NULL, NULL, NULL, &t,
NULL, NULL, NULL, NULL, NULL, NULL);
00712
00713
00714
           atm \rightarrow p[ip] = 1000. * pow(iso[ip] / t, -1. / 0.286);
00715
00716
00717
         /* Interpolate pressure... */
        else if (ctl->isosurf == 4) {
  if (atm->time[ip] <= ts[0])</pre>
00718
00719
             atm->p[ip] = ps[0];
00720
00721
           else if (atm->time[ip] >= ts[n - 1])
00722
             atm->p[ip] = ps[n - 1];
00723
           else {
             idx = locate_irr(ts, n, atm->time[ip]);
00724
             atm->p[ip] = LIN(ts[idx], ps[idx],
ts[idx + 1], ps[idx + 1], atm->time[ip]);
00725
00726
00727
00728
00729 }
```

Here is the call graph for this function:



5.35.2.6 void module_meteo ($ctl_t * ctl$, $met_t * met0$, $met_t * met1$, $atm_t * atm$, int ip)

Interpolate meteorological data for air parcel positions.

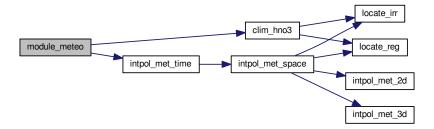
Definition at line 733 of file trac.c.

```
00738
                 {
00739
00740
        double a, b, c, ps, pt, pv, p_hno3, p_h2o, t, u, v, w, x1, x2, h2o, o3, z;
00741
00742
        /* Interpolate meteorological data... */
00743
        intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00744
                          atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &pv, &h2o, &o3);
00745
00746
         /* Set surface pressure... */
00747
        if (ctl->qnt_ps >= 0)
00748
          atm->q[ctl->qnt_ps][ip] = ps;
00749
00750
        /* Set tropopause pressure... */
00751
        if (ctl->qnt_pt >= 0)
00752
          atm->q[ctl->qnt_pt][ip] = pt;
00753
00754
        /* Set pressure... */
        if (ctl->qnt_p >= 0)
00755
00756
          atm->q[ctl->qnt_p][ip] = atm->p[ip];
00757
00758
        /* Set geopotential height... */
        if (ctl->qnt_z >= 0)
00759
00760
          atm \rightarrow q[ctl \rightarrow qnt_z][ip] = z;
00761
00762
        /* Set temperature... */
00763
        if (ctl->qnt_t >= 0)
00764
          atm \rightarrow q[ctl \rightarrow qnt_t][ip] = t;
00765
00766
        /* Set zonal wind... */
00767
        if (ctl->ant u >= 0)
00768
          atm->q[ctl->qnt_u][ip] = u;
00769
        /\star Set meridional wind... \star/
00770
00771
        if (ctl->qnt_v >= 0)
00772
          atm->q[ctl->qnt_v][ip] = v;
00773
00774
        /* Set vertical velocity... */
00775
        if (ctl->qnt_w >= 0)
00776
          atm \rightarrow q[ctl \rightarrow qnt_w][ip] = w;
00777
00778
        /* Set water vapor vmr... */
00779
        if (ctl->qnt_h2o >= 0)
          atm->q[ctl->qnt_h2o][ip] = h2o;
00780
00781
00782
        /* Set ozone vmr...
00783
        if (ctl->qnt_o3 >= 0)
00784
          atm \rightarrow q[ctl \rightarrow qnt_o3][ip] = o3;
00785
00786
        /* Calculate horizontal wind... */
00787
        if (ctl->qnt_vh >= 0)
00788
          atm \rightarrow q[ctl \rightarrow qnt\_vh][ip] = sqrt(u * u + v * v);
00789
00790
         /* Calculate vertical velocity... */
00791
        if (ctl->ant vz >= 0)
00792
          atm \rightarrow q[ctl \rightarrow qnt_vz][ip] = -1e3 * H0 / atm \rightarrow p[ip] * w;
00793
00794
        /* Calculate potential temperature... */
00795
        if (ctl->qnt_theta >= 0)
00796
          atm->q[ctl->qnt_theta][ip] = THETA(atm->p[ip], t);
00797
00798
        /* Set potential vorticity... */
00799
        if (ctl->qnt_pv >= 0)
00800
          atm->q[ctl->qnt_pv][ip] = pv;
00801
00802
         /* Calculate T_ice (Marti and Mauersberger, 1993)... */
        if (ctl->qnt_tice >= 0)
00803
          atm->q[ctl->qnt_tice][ip] =
00804
00805
             -2663.5 /
00806
             (log10((ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] * 100.) -
00807
00808
00809
         /\star Calculate T_NAT (Hanson and Mauersberger, 1988)... \star/
        if (ctl->qnt_tnat >= 0) {
   if (ctl->psc_hno3 > 0)
00810
00811
00812
            p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
00813
00814
           p_hno3 = clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip])
          * 1e-9 * atm->p[ip] / 1.333224;

p_h2o = (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
00815
00816
          b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
00817
00818
00819
           c = -11397.0 / a;
00820
           x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
           x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
00821
           if (x1 > 0)
00822
00823
             atm->g[ctl->gnt tnat][ip] = x1;
```

```
if (x2 > 0)
             atm->q[ctl->qnt_tnat][ip] = x2;
00826
00827
00828
         /* Calculate T_STS (mean of T_ice and T_NAT)... */
         if (ctl->qnt_tsts >= 0) {
   if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)</pre>
00829
00831
             ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
00832
           atm -> q[ctl -> qnt\_tsts][ip] = 0.5 * (atm -> q[ctl -> qnt\_tice][ip]
00833
                                                   + atm->q[ctl->qnt_tnat][ip]);
00834
         }
00835 }
```

Here is the call graph for this function:



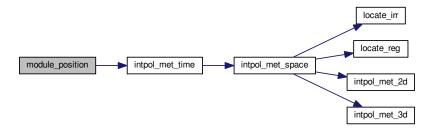
5.35.2.7 void module_position ($met_t * met0$, $met_t * met1$, $atm_t * atm$, int ip)

Check position of air parcels.

Definition at line 839 of file trac.c.

```
00843
00844
00845
          double ps;
00846
00847
          /* Calculate modulo... */
00848
          atm \rightarrow lon[ip] = fmod(atm \rightarrow lon[ip], 360);
00849
          atm \rightarrow lat[ip] = fmod(atm \rightarrow lat[ip], 360);
00850
          /* Check latitude... */
while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
   if (atm->lat[ip] > 90) {
     atm->lat[ip] = 180 - atm->lat[ip];
00851
00852
00853
00854
00855
                atm->lon[ip] += 180;
00856
             if (atm->lat[ip] < -90) {
  atm->lat[ip] = -180 - atm->lat[ip];
  atm->lon[ip] += 180;
00857
00858
00859
00860
00861
00862
          /* Check longitude... */
while (atm->lon[ip] < -180)
  atm->lon[ip] += 360;
while (atm->lon[ip] >= 180)
  atm->lon[ip] -= 360;
00863
00864
00865
00866
00867
00868
00869
          /* Get surface pressure... */
          00870
00871
                                 NULL, NULL, NULL, NULL, NULL, NULL);
00872
00873
00874
           /* Check pressure... */
          if (atm->p[ip] > ps)
  atm->p[ip] = ps;
else if (atm->p[ip] < met0->p[met0->np - 1])
00875
00876
00877
             atm->p[ip] = met0->p[met0->np - 1];
00878
00879 }
```

Here is the call graph for this function:



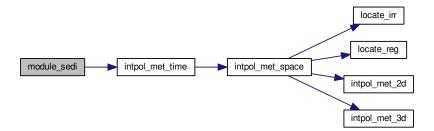
5.35.2.8 void module_sedi ($ctl_t * ctl$, $met_t * met0$, $met_t * met1$, $atm_t * atm$, int ip, double dt)

Calculate sedimentation of air parcels.

Definition at line 883 of file trac.c.

```
00889
00890
00891
        /\star Coefficients for Cunningham slip-flow correction (Kasten, 1968): \star/
00892
        const double A = 1.249, B = 0.42, C = 0.87;
00893
00894
        /* Average mass of an air molecule [kg/molec]: */
00895
        const double m = 4.8096e-26;
00896
00897
        double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
00898
00899
        /* Convert units... */
00900
        p = 100 * atm->p[ip];
00901
        r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
00902
        rho_p = atm->q[ctl->qnt_rho][ip];
00903
00904
        /* Get temperature... */
00905
       intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00906
                         atm->lat[ip], NULL, NULL, NULL, &T,
00907
                         NULL, NULL, NULL, NULL, NULL, NULL);
00908
       /* Density of dry air... */ rho = p / (RA * T);
00909
00910
00911
        /* Dynamic viscosity of air... */
00912
00913
        eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
00914
00915
        /\star Thermal velocity of an air molecule... \star/
00916
        v = sqrt(8 * KB * T / (M_PI * m));
00917
        /\star Mean free path of an air molecule... \star/
00918
        lambda = 2 * eta / (rho * v);
00919
00920
00921
        /\star Knudsen number for air... \star/
00922
        K = lambda / r_p;
00923
00924
        /* Cunningham slip-flow correction... */
00925
        G = 1 + K * (A + B * exp(-C / K));
00926
00927
        /* Sedimentation (fall) velocity... */
        v_p = 2. * SQR(r_p) * (rho_p - rho) * GO / (9. * eta) * G;
00928
00929
00930
        /* Calculate pressure change... */
00931
       atm->p[ip] += DZ2DP(v_p * dt / 1000., atm->p[ip]);
00932 }
```

Here is the call graph for this function:



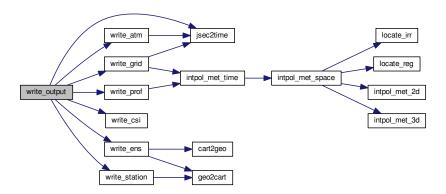
5.35.2.9 void write_output (const char * dirname, ctl t * ctl, met t * met0, met t * met1, atm t * atm, double t)

Write simulation output.

Definition at line 936 of file trac.c.

```
00942
                     {
00943
00944
         char filename[2 * LEN]:
00945
00946
00947
00948
         int year, mon, day, hour, min, sec;
00949
00950
         /* Get time... */
00951
         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
00952
         /* Write atmospheric data... */
if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
    sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
00953
00954
00955
00956
                    dirname, ctl->atm_basename, year, mon, day, hour, min);
00957
           write_atm(filename, ctl, atm, t);
00958
00959
00960
         /* Write CSI data... */
         if (ctl->csi_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
00961
00962
00963
           write_csi(filename, ctl, atm, t);
00964
00965
         /* Write ensemble data... */
if (ctl->ens_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
00966
00967
00968
           write_ens(filename, ctl, atm, t);
00969
00970
00971
00972
         /* Write gridded data...
         if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
00973
           00974
00975
00976
00977
00978
         /* Write profile data... */
if (ctl->prof_basename[0] != '-') {
00979
00980
         sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
00981
00982
           write_prof(filename, ctl, met0, met1, atm, t);
00983
00984
00985
         /* Write station data...
         /^ wille station data... '/
if (ctl->stat_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
00986
00987
00988
           write_station(filename, ctl, atm, t);
00989
00990 }
```

Here is the call graph for this function:



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

Definition at line 115 of file trac.c.

```
00117
00118
00119
        ctl_t ctl;
00120
00121
       atm_t *atm;
00122
00123
       met_t *met0, *met1;
00124
       gsl_rng *rng[NTHREADS];
00125
00126
00127
       FILE *dirlist;
00128
00129
       char dirname[LEN], filename[2 * LEN];
00130
00131
       double *dt, t;
00132
00133
        int i, ip, ntask = -1, rank = 0, size = 1;
00134
00135 #ifdef MPI
00136
        /* Initialize MPI... */
00137
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00138
00139
        MPI_Comm_size(MPI_COMM_WORLD, &size);
00140 #endif
00141
00142
        /* Check arguments... */
        if (argc < 5)
00143
00144
          ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00145
00146
        /* Open directory list... */
00147
        if (!(dirlist = fopen(argv[1], "r")))
00148
          ERRMSG("Cannot open directory list!");
00149
        /* Loop over directories... */
while (fscanf(dirlist, "%s", dirname) != EOF) {
00150
00151
00152
          /* MPI parallelization... */
00153
00154
          if ((++ntask) % size != rank)
00155
            continue;
00156
00157
00158
            Initialize model run...
00159
00160
00161
          /* Set timers...
          START_TIMER (TIMER_TOTAL);
00162
00163
          START_TIMER(TIMER_INIT);
00164
00165
          /* Allocate... */
00166
          ALLOC(atm, atm_t, 1);
```

```
00167
          ALLOC(met0, met_t, 1);
          ALLOC(met1, met_t, 1);
00168
00169
          ALLOC(dt, double,
00170
                 NP);
00171
00172
          /* Initialize random number generators... */
00173
          gsl_rng_env_setup();
00174
              (omp_get_max_threads() > NTHREADS)
00175
            ERRMSG("Too many threads!");
00176
           for (i = 0; i < NTHREADS; i++) {</pre>
            rng[i] = gsl_rng_alloc(gsl_rng_default);
00177
            gsl_rng_set(rng[i], gsl_rng_default_seed + (long unsigned) i);
00178
00179
00180
00181
          /\star Read control parameters... \star/
00182
          sprintf(filename, "%s/%s", dirname, argv[2]);
00183
          read_ctl(filename, argc, argv, &ctl);
00184
00185
          /* Read atmospheric data... */
          sprintf(filename, "%s/%s", dirname, argv[3]);
00186
00187
          read_atm(filename, &ctl, atm);
00188
00189
           /* Set start time... */
          if (ctl.direction == 1) {
00190
00191
            ctl.t_start = qsl_stats_min(atm->time, 1, (size_t) atm->np);
00192
             if (ctl.t_stop > 1e99)
00193
               ctl.t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
          } else {
00194
00195
             ctl.t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
             if (ctl.t_stop > 1e99)
  ctl.t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00196
00197
00198
00199
00200
          /* Check time interval... */
00201
          if (ctl.direction * (ctl.t_stop - ctl.t_start) <= 0)</pre>
            ERRMSG("Nothing to do!");
00202
00203
00204
           /* Round start time... */
00205
          if (ctl.direction == 1)
            ctl.t_start = floor(ctl.t_start / ctl.dt_mod) * ctl.
00206
      dt_mod;
00207
          else
            ctl.t start = ceil(ctl.t start / ctl.dt mod) * ctl.
00208
      dt_mod;
00209
00210
           /* Set timers...
00211
          STOP_TIMER(TIMER_INIT);
00212
00213
00214
            Loop over timesteps...
00215
00216
00217
          /* Loop over timesteps... */
00218
          for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.</pre>
      dt_mod;
00219
                t += ctl.direction * ctl.dt mod) {
00220
00221
             /* Adjust length of final time step... */
00222
            if (ctl.direction * (t - ctl.t_stop) > 0)
00223
               t = ctl.t_stop;
00224
00225
             /\star Set time steps for air parcels... \star/
00226
            for (ip = 0; ip < atm->np; ip++)
              if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
00227
                    00228
00229
00230
                 dt[ip] = t - atm->time[ip];
00231
              else
00232
                 dt[ip] = GSL_NAN;
00234
             /* Get meteorological data... */
00235
             START_TIMER(TIMER_INPUT);
             get_met(&ctl, argv[4], t, &met0, &met1);
if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
   printf("Warning: Violation of CFL criterion! Set DT_MOD <= %g s!\n",</pre>
00236
00237
00238
00239
                      fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.);
00240
            STOP_TIMER(TIMER_INPUT);
00241
             /* Initialize isosurface... */
00242
             START_TIMER(TIMER_ISOSURF);
00243
            if (ctl.isosurf >= 1 && ctl.isosurf <= 4 && t == ctl.t_start)
module_isosurf(&ctl, met0, met1, atm, -1);</pre>
00244
00245
00246
             STOP_TIMER(TIMER_ISOSURF);
00247
             /* Advection... */
00248
            START_TIMER (TIMER_ADVECT);
00249
00250 #pragma omp parallel for default(shared) private(ip)
```

```
for (ip = 0; ip < atm->np; ip++)
            if (gsl_finite(dt[ip]))
00252
00253
               module_advection(met0, met1, atm, ip, dt[ip]);
00254
            STOP_TIMER(TIMER_ADVECT);
00255
00256
            /* Turbulent diffusion... */
            START_TIMER(TIMER_DIFFTURB);
00258
            if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
00259
                || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0) {
00260 #pragma omp parallel for default(shared) private(ip)
             for (ip = 0; ip < atm->np; ip++)
00261
00262
               if (gsl_finite(dt[ip]))
00263
                 module_diffusion_turb(&ctl, atm, ip, dt[ip],
00264
                                        rng[omp_get_thread_num()]);
00265
00266
            STOP_TIMER(TIMER_DIFFTURB);
00267
00268
            /* Mesoscale diffusion...
            START_TIMER(TIMER_DIFFMESO);
00269
00270
            if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0) {
00271 #pragma omp parallel for default(shared) private(ip)
00272
             for (ip = 0; ip < atm->np; ip++)
               if (gsl_finite(dt[ip]))
00273
00274
                 module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
00275
                                        rng[omp_get_thread_num()]);
00276
00277
            STOP_TIMER(TIMER_DIFFMESO);
00278
            /* Sedimentation...
00279
            START_TIMER(TIMER_SEDI);
00280
00281
            if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0) {
00282 #pragma omp parallel for default(shared) private(ip)
00283
            for (ip = 0; ip < atm->np; ip++)
00284
               if (gsl_finite(dt[ip]))
00285
                 module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
00286
00287
            STOP_TIMER(TIMER_SEDI);
00289
            /* Isosurface..
00290
            START_TIMER(TIMER_ISOSURF);
00291
            if (ctl.isosurf >= 1 && ctl.isosurf <= 4) {</pre>
00292 \#pragma omp parallel for default(shared) private(ip)
             for (ip = 0; ip < atm->np; ip++)
00293
               module_isosurf(&ctl, met0, met1, atm, ip);
00294
00295
00296
            STOP_TIMER(TIMER_ISOSURF);
00297
00298
            /* Position...
           START_TIMER(TIMER_POSITION);
00299
00300 #pragma omp parallel for default(shared) private(ip)
           for (ip = 0; ip < atm->np; ip++)
00301
00302
             module_position(met0, met1, atm, ip);
00303
            STOP_TIMER(TIMER_POSITION);
00304
            /* Meteorological data... */
00305
00306
            START_TIMER(TIMER_METEO);
            if (ctl.met_dt_out > 0
                && (ctl.met_dt_out < ctl.dt_mod || fmod(t, ctl.
00308
     met_dt_out) == 0)) {
00309 \#pragma omp parallel for default(shared) private(ip)
             for (ip = 0; ip < atm->np; ip++)
00310
00311
               module_meteo(&ctl, met0, met1, atm, ip);
00312
00313
            STOP_TIMER(TIMER_METEO);
00314
            /* Decay... */
00315
00316
            START_TIMER(TIMER_DECAY);
            if ((ctl.tdec_trop > 0 || ctl.tdec_strat > 0) && ctl.
00317
      ant m \ge 0) {
00318 #pragma omp parallel for default(shared) private(ip)
00319
            for (ip = 0; ip < atm->np; ip++)
00320
                if (gsl_finite(dt[ip]))
00321
                 module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00322
00323
            STOP_TIMER(TIMER_DECAY);
00324
00325
            /* Write output... */
00326
            START_TIMER(TIMER_OUTPUT);
00327
            write_output(dirname, &ctl, met0, met1, atm, t);
            STOP_TIMER(TIMER_OUTPUT);
00328
00329
00330
00331
00332
            Finalize model run...
00333
00334
00335
          /* Report memory usage... */
```

```
printf("MEMORY_ATM = %g MByte\n", sizeof(atm_t) / 1024. / 1024.);
printf("MEMORY_METEO = %g MByte\n", 2. * sizeof(met_t) / 1024. / 1024.);
printf("MEMORY_DYNAMIC = %g MByte\n",
00337
00338
               ### 4 * NP * sizeof(double) / 1024. / 1024.);

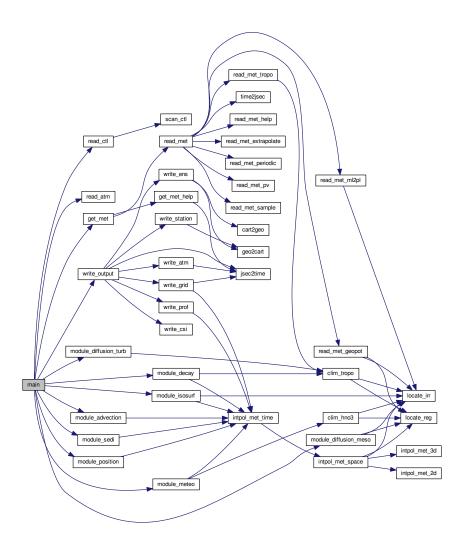
printf("MEMORY_STATIC = %g MByte\n",

((3 * GX * GY + 4 * GX * GY * GZ) * sizeof(double)

+ (EX * EY + EX * EY * EP) * sizeof(float)

+ (GX * GY + GX * GY * GZ) * sizeof(int)) / 1024. / 1024.);
00339
00340
00341
00342
00343
00344
               /* Report problem size... */
printf("SIZE_NP = %d\n", atm->np);
printf("SIZE_TASKS = %d\n", size);
00345
00346
00347
               printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00348
00349
00350
                /* Report timers...
00351
               STOP_TIMER(TIMER_TOTAL);
00352
               PRINT_TIMER(TIMER_TOTAL);
               PRINT_TIMER(TIMER_INIT);
PRINT_TIMER(TIMER_INPUT);
00353
00354
00355
               PRINT_TIMER (TIMER_OUTPUT);
00356
               PRINT_TIMER (TIMER_ADVECT);
00357
               PRINT_TIMER(TIMER_DECAY);
00358
               PRINT_TIMER(TIMER_DIFFMESO);
               PRINT_TIMER (TIMER_DIFFTURB);
PRINT_TIMER (TIMER_ISOSURF);
PRINT_TIMER (TIMER_METEO);
00359
00360
00361
00362
               PRINT_TIMER(TIMER_POSITION);
00363
               PRINT_TIMER(TIMER_SEDI);
00364
00365
               /* Free random number generators... */
for (i = 0; i < NTHREADS; i++)</pre>
00366
00367
                 gsl_rng_free(rng[i]);
00368
               /* Free... */
00369
00370
               free(atm);
00371
               free (met0);
00372
               free(met1);
00373
               free(dt);
00374
00375
00376 #ifdef MPI
00377 /* Finalize MPI... */
00378 MPI_Finalize();
00379 #endif
00380
00381
            return EXIT_SUCCESS;
00382 }
```

Here is the call graph for this function:



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

```
00032
         Functions...
00033
00034
00036 void module advection(
        met_t * met0,
met_t * met1,
00037
00039
         atm_t * atm,
00040
        int ip,
00041
        double dt);
00042
00044 void module_decay(
        ctl_t * ctl,
met_t * met0,
00045
00046
00047
         met_t * met1,
00048
         atm_t * atm,
00049
        int ip,
00050
        double dt);
00051
00053 void module_diffusion_meso(
        ctl_t * ctl,
met_t * met0,
00054
00055
        met_t * met1,
atm_t * atm,
00056
00057
00058
        int ip,
00059
        double dt,
00060
        gsl_rng * rng);
00061
00063 void module_diffusion_turb(
00064
        ctl_t * ctl,
atm_t * atm,
00065
00066
        int ip,
00067
        double dt,
00068
        gsl_rng * rng);
00069
00071 void module_isosurf(
00072
        ctl_t * ctl,
met_t * met0,
00073
        met_t * met1,
atm_t * atm,
00074
00075
00076
        int ip);
00077
00079 void module_meteo(
        ctl_t * ctl,
met_t * met0,
08000
00081
00082
        met_t * met1,
00083
        atm_t * atm,
00084
        int ip);
00085
00087 void module_position(
        met_t * met0,
met_t * met1,
atm_t * atm,
00088
00089
00090
00091
        int ip);
00092
00094 void module_sedi(
        ctl_t * ctl,
met_t * met0,
00095
00096
00097
         met_t * met1,
        atm_t * atm,
00098
00099
        int ip,
00100
        double dt);
00101
00103 void write_output(
00104
        const char *dirname,
        ctl_t * ctl,
met_t * met0,
00105
00106
        met_t * met1,
atm_t * atm,
00107
00108
00109
         double t);
00110
00111 /* -----
00112
          Main...
00113
00114
00115 int main(
00116
        int argc,
00117
        char *argv[]) {
00118
00119
        ctl t ctl;
00120
00121
        atm_t *atm;
00122
00123
        met_t *met0, *met1;
00124
         gsl_rng *rng[NTHREADS];
00125
00126
```

```
00127
        FILE *dirlist;
00128
00129
        char dirname[LEN], filename[2 * LEN];
00130
00131
        double *dt. t:
00132
        int i, ip, ntask = -1, rank = 0, size = 1;
00133
00134
00135 #ifdef MPI
00136
        /* Initialize MPI... */
00137
        MPI_Init(&argc, &argv);
        MPI_Comm_rank (MPI_COMM_WORLD, &rank);
00138
00139
        MPI_Comm_size(MPI_COMM_WORLD, &size);
00140 #endif
00141
00142
         /* Check arguments... */
        if (argc < 5)
00143
          ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00144
00145
00146
        /* Open directory list... */
00147
        if (!(dirlist = fopen(argv[1], "r")))
00148
          ERRMSG("Cannot open directory list!");
00149
        /* Loop over directories... */
while (fscanf(dirlist, "%s", dirname) != EOF) {
00150
00151
00152
00153
           /* MPI parallelization... */
00154
          if ((++ntask) % size != rank)
00155
             continue;
00156
00157
00158
              Initialize model run...
00159
00160
           /* Set timers... */
START_TIMER(TIMER_TOTAL);
00161
00162
           START_TIMER(TIMER_INIT);
00163
00164
00165
           /* Allocate... */
00166
           ALLOC(atm, atm_t, 1);
00167
           ALLOC(met0, met_t, 1);
           ALLOC(met1, met_t, 1);
00168
           ALLOC(dt, double, NP);
00169
00170
00171
00172
           /* Initialize random number generators... */
00173
           gsl_rng_env_setup();
00174
           if (omp_get_max_threads() > NTHREADS)
             ERRMSG("Too many threads!");
00175
           for (i = 0; i < NTHREADS; i++) {
00176
            rng[i] = gsl_rng_alloc(gsl_rng_default);
00178
             gsl_rng_set(rng[i], gsl_rng_default_seed + (long unsigned) i);
00179
00180
           /* Read control parameters... */
sprintf(filename, "%s/%s", dirname, argv[2]);
read_ctl(filename, argc, argv, &ctl);
00181
00182
00184
           /* Read atmospheric data... */
sprintf(filename, "%s/%s", dirname, argv[3]);
read_atm(filename, &ctl, atm);
00185
00186
00187
00188
00189
           /* Set start time... *,
00190
           if (ctl.direction == 1) {
00191
             ctl.t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
             if (ctl.t_stop > 1e99)
  ctl.t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00192
00193
           } else {
00194
00195
             ctl.t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
             if (ctl.t_stop > 1e99)
00196
00197
               ctl.t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00198
00199
           /* Check time interval... */
if (ctl.direction * (ctl.t_stop - ctl.t_start) <= 0)</pre>
00200
00201
00202
             ERRMSG("Nothing to do!");
00203
           /* Round start time...
00204
00205
           if (ctl.direction == 1)
            ctl.t_start = floor(ctl.t_start / ctl.dt_mod) * ctl.
00206
      dt mod;
00207
           else
             ctl.t_start = ceil(ctl.t_start / ctl.dt_mod) * ctl.
      dt_mod;
00209
           /* Set timers... */
00210
00211
           STOP_TIMER(TIMER_INIT);
```

```
00213
00214
             Loop over timesteps...
00215
00216
00217
          /* Loop over timesteps... */
          for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.</pre>
00218
00219
               t += ctl.direction * ctl.dt_mod) {
00220
00221
            /* Adjust length of final time step... */
            if (ctl.direction * (t - ctl.t_stop) > 0)
00222
00223
              t = ctl.t stop;
00224
00225
            /\star Set time steps for air parcels... \star/
00226
            for (ip = 0; ip < atm->np; ip++)
              if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
00227
                   && ctl.direction * (atm->time[ip] - ctl.t_stop) <= 0
&& ctl.direction * (atm->time[ip] - t) < 0))
00228
00230
                dt[ip] = t - atm->time[ip];
00231
00232
                dt[ip] = GSL_NAN;
00233
00234
            /* Get meteorological data... */
00235
            START_TIMER(TIMER_INPUT);
00236
            get_met(&ctl, argv[4], t, &met0, &met1);
00237
            if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00238
             printf("Warning: Violation of CFL criterion! Set DT_MOD <= %g s!\n",
00239
                     fabs (met 0 - > lon[1] - met 0 - > lon[0]) * 111132. / 150.);
            STOP_TIMER(TIMER_INPUT);
00240
00241
00242
             /* Initialize isosurface... */
00243
            START_TIMER(TIMER_ISOSURF);
00244
            if (ctl.isosurf >= 1 && ctl.isosurf <= 4 && t == ctl.t_start)</pre>
              module_isosurf(&ctl, met0, met1, atm, -1);
00245
00246
            STOP_TIMER(TIMER_ISOSURF);
00247
            /* Advection...
00249
            START_TIMER(TIMER_ADVECT);
00250 #pragma omp parallel for default(shared) private(ip)
00251
            for (ip = 0; ip < atm->np; ip++)
             if (gsl_finite(dt[ip]))
00252
                module advection(met0, met1, atm, ip, dt[ip]);
00253
00254
            STOP_TIMER(TIMER_ADVECT);
00255
00256
            /* Turbulent diffusion..
00257
            START_TIMER(TIMER_DIFFTURB);
            00258
00259
00260 #pragma omp parallel for default(shared) private(ip)
             for (ip = 0; ip < atm->np; ip++)
               if (gsl_finite(dt[ip]))
00262
00263
                  module_diffusion_turb(&ctl, atm, ip, dt[ip],
00264
                                         rng[omp_get_thread_num()]);
00265
00266
            STOP TIMER (TIMER DIFFTURB);
00268
             /* Mesoscale diffusion...
00269
            START_TIMER(TIMER_DIFFMESO);
00270
            if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0) {
00271 #pragma omp parallel for default(shared) private(ip)
             for (ip = 0; ip < atm->np; ip++)
00272
00273
                if (gsl_finite(dt[ip]))
00274
                  module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
00275
                                         rng[omp_get_thread_num()]);
00276
00277
            STOP TIMER (TIMER DIFFMESO);
00278
00279
             /* Sedimentation...
            START_TIMER(TIMER_SEDI);
00281
            if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0) {
00282 #pragma omp parallel for default(shared) private(ip)
             for (ip = 0; ip < atm->np; ip++)
  if (gsl_finite(dt[ip]))
00283
00284
00285
                  module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
00286
00287
            STOP_TIMER(TIMER_SEDI);
00288
00289
            /* Isosurface... */
            START TIMER (TIMER ISOSURF):
00290
            if (ctl.isosurf >= 1 && ctl.isosurf <= 4) {</pre>
00291
00292 #pragma omp parallel for default(shared) private(ip)
00293
             for (ip = 0; ip < atm->np; ip++)
00294
                module_isosurf(&ctl, met0, met1, atm, ip);
00295
00296
            STOP TIMER (TIMER ISOSURF);
00297
```

```
00298
            /* Position...
00299
            START_TIMER(TIMER_POSITION);
00300 #pragma omp parallel for default(shared) private(ip)
00301
           for (ip = 0; ip < atm->np; ip++)
              module_position(met0, met1, atm, ip);
00302
00303
            STOP_TIMER(TIMER_POSITION);
00305
            /* Meteorological data... */
00306
            START_TIMER(TIMER_METEO);
00307
            if (ctl.met_dt_out > 0
                && (ctl.met_dt_out < ctl.dt_mod || fmod(t, ctl.
00308
met_dt_out) == 0)) {
00309 #pragma omp parallel for default(shared) private(ip)
00310
             for (ip = 0; ip < atm->np; ip++)
00311
                module_meteo(&ctl, met0, met1, atm, ip);
00312
            STOP TIMER (TIMER METEO);
00313
00314
            /* Decay... */
00316
            START_TIMER(TIMER_DECAY);
            if ((ctl.tdec_trop > 0 || ctl.tdec_strat > 0) && ctl.
00317
      qnt_m >= 0) {
00318 #pragma omp parallel for default(shared) private(ip)
00319 for (ip = 0; ip < atm->np; ip++)
00320
                if (gsl_finite(dt[ip]))
00321
                 module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00322
00323
            STOP_TIMER(TIMER_DECAY);
00324
            /* Write output... */
00325
            START_TIMER (TIMER_OUTPUT);
00326
00327
            write_output(dirname, &ctl, met0, met1, atm, t);
00328
            STOP_TIMER(TIMER_OUTPUT);
00329
00330
00331
00332
             Finalize model run...
00333
00334
00335
          /* Report memory usage... */
          00336
00337
00338
00339
00340
          printf("MEMORY_STATIC = %g MByte\n",
                 ((3 * GX * GY + 4 * GX * GY * GZ) * sizeof(double)
00341
                  + (EX * EY + EX * EY * EP) * sizeof(float)
+ (GX * GY + GX * GY * GZ) * sizeof(int)) / 1024. / 1024.);
00342
00343
00344
00345
          /* Report problem size... */
          printf("SIZE_NP = %d\n", atm->np);
00346
00347
          printf("SIZE_TASKS = %d\n", size);
          printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00348
00349
00350
          /* Report timers... */
00351
          STOP_TIMER(TIMER_TOTAL);
          PRINT_TIMER (TIMER_TOTAL);
00353
          PRINT_TIMER(TIMER_INIT);
00354
          PRINT_TIMER (TIMER_INPUT);
00355
          PRINT_TIMER (TIMER_OUTPUT);
          PRINT_TIMER(TIMER_ADVECT);
00356
          PRINT_TIMER(TIMER_DECAY);
00357
00358
          PRINT_TIMER (TIMER_DIFFMESO);
00359
          PRINT_TIMER(TIMER_DIFFTURB);
00360
          PRINT_TIMER(TIMER_ISOSURF);
00361
          PRINT_TIMER(TIMER_METEO);
00362
          PRINT TIMER (TIMER POSITION);
          PRINT_TIMER(TIMER_SEDI);
00363
00364
00365
          /* Free random number generators... */
00366
          for (i = 0; i < NTHREADS; i++)</pre>
00367
            gsl_rng_free(rng[i]);
00368
          /* Free... */
00369
00370
          free (atm);
00371
          free (met0);
00372
          free (met1);
00373
          free(dt);
00374
00375
00376 #ifdef MPI
        /* Finalize MPI... */
00378
       MPI_Finalize();
00379 #endif
00380
00381
        return EXIT_SUCCESS;
00382 }
```

```
00385
00386 void module_advection(
00387
      met_t * met0,
met_t * met1,
00388
       atm_t * atm,
00389
00390
       int ip,
00391
       double dt) {
00392
       double v[3], xm[3];
00393
00394
00395
       /* Interpolate meteorological data... */
00396
       intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00397
                     atm->lon[ip], atm->lat[ip], NULL, NULL, NULL, NULL,
00398
                      &v[0], &v[1], &v[2], NULL, NULL, NULL);
00399
00400
       /* Get position of the mid point... */
       xm[0] = atm->lon[ip] + DX2DEG(0.5 * dt * v[0] / 1000., atm->lat[ip]);
xm[1] = atm->lat[ip] + DY2DEG(0.5 * dt * v[1] / 1000.);
00401
00402
00403
       xm[2] = atm - p[ip] + 0.5 * dt * v[2];
00404
00405
       /* Interpolate meteorological data for mid point... */
       00406
00407
00408
00409
00410
       /* Save new position... */
00411
       atm->time[ip] += dt;
       atm->lon[ip] += DX2DEG(dt * v[0] / 1000., xm[1]);
atm->lat[ip] += DY2DEG(dt * v[1] / 1000.);
00412
00413
00414
       atm \rightarrow p[ip] += dt * v[2];
00415 }
00416
00418
00419 void module_decay(
00420
      ctl_t * ctl,
00421
       met_t * met0,
00422
00423
       atm_t * atm,
00424
       int ip,
00425
       double dt) {
00426
00427
       double ps, pt, tdec;
00428
00429
       /* Set constant lifetime... */
00430
       if (ctl->tdec_trop == ctl->tdec_strat)
        tdec = ctl->tdec_trop;
00431
00432
00433
       /* Set altitude-dependent lifetime... */
00434
00435
        00436
00437
00438
00440
00441
         /* Get tropopause pressure... */
00442
        pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00443
         /* Set lifetime... */
00444
00445
         if (atm->p[ip] <= pt)</pre>
00446
          tdec = ctl->tdec_strat;
00447
00448
          tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
     p[ip]);
00449
00450
00451
       /* Calculate exponential decay... */
00452
       atm \rightarrow q[ctl \rightarrow qnt_m][ip] *= exp(-dt / tdec);
00453 }
00454
00456
00457 void module_diffusion_meso(
00458
       ctl_t * ctl,
00459
       met_t * met0,
00460
       met_t * met1,
       atm_t * atm,
00461
00462
       int ip,
00463
       double dt,
00464
       gsl_rng * rng) {
00465
00466
      double r, rs, u[16], v[16], w[16];
00467
00468
      int ix, iv, iz:
```

```
00469
00470
          /* Get indices... */
00471
         ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
         iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
00472
00473
         iz = locate_irr(met0->p, met0->np, atm->p[ip]);
00474
00475
          /* Caching of wind standard deviations... */
00476
          if (atm->cache_time[ix][iy][iz] != met0->time) {
00477
00478
            /* Collect local wind data... */
           u[0] = met0->u[ix][iy][iz];
u[1] = met0->u[ix + 1][iy][iz];
00479
00480
            u[2] = met0->u[ix][iy + 1][iz];
00481
            u[3] = met0 - u[ix + 1][iy + 1][iz];
00482
            u[4] = met0->u[ix][iy][iz + 1];
00483
           u[5] = met0->u[ix + 1][iy][iz + 1];

u[6] = met0->u[ix][iy + 1][iz + 1];
00484
00485
            u[7] = met0 -> u[ix + 1][iy + 1][iz + 1];
00486
00488
            v[0] = met0 -> v[ix][iy][iz];
00489
            v[1] = met0 -> v[ix + 1][iy][iz];
00490
            v[2] = met0 -> v[ix][iy + 1][iz];
           v(z) = met0->v[ix]|iy + 1][iz];
v(3] = met0->v[ix + 1][iy + 1][iz];
v[4] = met0->v[ix][iy][iz + 1];
v[5] = met0->v[ix + 1][iy][iz + 1];
v[6] = met0->v[ix][iy + 1][iz + 1];
00491
00492
00493
00494
00495
            v[7] = met0 -> v[ix + 1][iy + 1][iz + 1];
00496
00497
            w[0] = met0->w[ix][iy][iz];
           w[1] = met0->w[ix + 1][iy][iz];
w[2] = met0->w[ix][iy + 1][iz];
00498
00499
            w[3] = met0->w[ix + 1][iy + 1][iz];
w[4] = met0->w[ix][iy][iz + 1];
00500
00501
00502
            w[5] = met0->w[ix + 1][iy][iz + 1];
            w[6] = met0->w[ix][iy + 1][iz + 1];

w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00503
00504
00505
            /* Collect local wind data... */
00507
            u[8] = met1->u[ix][iy][iz];
00508
            u[9] = met1 -> u[ix + 1][iy][iz];
00509
            u[10] = met1->u[ix][iy + 1][iz];
           u[11] = met1->u[ix + 1][iy + 1][iz];
u[12] = met1->u[ix][iy][iz + 1];
00510
00511
00512
            u[13] = met1 -> u[ix + 1][iy][iz + 1];
            u[14] = met1->u[ix][iy + 1][iz + 1];
00513
00514
            u[15] = met1 -> u[ix + 1][iy + 1][iz + 1];
00515
            v[8] = met1->v[ix][iy][iz];
v[9] = met1->v[ix + 1][iy][iz];
00516
00517
            v[10] = met1->v[ix][iy + 1][iz];
00518
            v[11] = met1->v[ix][iy + 1][iz];
v[11] = met1->v[ix + 1][iy + 1][iz];
v[12] = met1->v[ix][iy][iz + 1];
00520
00521
            v[13] = met1 -> v[ix + 1][iy][iz + 1];
            v[14] = met1->v[ix][iy + 1][iz + 1];
00522
00523
            v[15] = met1 - v[ix + 1][iy + 1][iz + 1];
00524
            w[8] = met1->w[ix][iy][iz];
            w[9] = met1->w[ix + 1][iy][iz];
00526
00527
            w[10] = met1->w[ix][iy + 1][iz];
            w[11] = met1->w[ix + 1][iy + 1][iz];
00528
            w[12] = met1->w[ix][iy][iz + 1];
00529
            w[13] = met1->w[ix + 1][iy][iz + 1];
00530
            w[14] = met1->w[ix][iy + 1][iz + 1];
w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00531
00532
00533
            /\star Get standard deviations of local wind data... \star/
00534
00535
            atm->cache_usig[ix][iy][iz] = (float) gsl_stats_sd(u, 1, 16);
atm->cache_vsig[ix][iy][iz] = (float) gsl_stats_sd(v, 1, 16);
00536
00537
            atm->cache_wsig[ix][iy][iz] = (float) gsl_stats_sd(w, 1, 16);
            atm->cache_time[ix][iy][iz] = met0->time;
00538
00539
00540
00541
         /\star Set temporal correlations for mesoscale fluctuations... \star/
         r = 1 - 2 * fabs(dt) / ctl->dt_met;
00542
00543
         rs = sqrt(1 - r * r);
00544
00545
          /* Calculate horizontal mesoscale wind fluctuations... */
00546
         if (ctl->turb_mesox > 0) {
00547
            atm->up[ip] = (float)
00548
              (r * atm->up[ip]
00549
                + rs * gsl_ran_gaussian_ziggurat(rng,
                                                        ctl->turb_mesox *
00551
                                                        atm->cache_usig[ix][iy][iz]));
00552
            atm->lon[ip] += DX2DEG(atm->up[ip] * dt / 1000., atm->lat[ip]);
00553
            atm->vp[ip] = (float)
00554
00555
              (r * atm->vp[ip]
```

```
+ rs * qsl_ran_gaussian_ziggurat(rng,
00557
                                            ctl->turb_mesox *
00558
                                            atm->cache_vsig[ix][iy][iz]));
         atm->lat[ip] += DY2DEG(atm->vp[ip] * dt / 1000.);
00559
00560
00561
00562
       /\star Calculate vertical mesoscale wind fluctuations... \star/
00563
       if (ctl->turb_mesoz > 0) {
00564
       atm->wp[ip] = (float)
00565
          (r * atm->wp[ip]
00566
            + rs * gsl_ran_gaussian_ziggurat(rng,
00567
                                            ctl->turb mesoz *
00568
                                            atm->cache wsig[ix][iv][iz]));
00569
         atm->p[ip] += atm->wp[ip] * dt;
00570
00571 }
00572
00574
00575 void module_diffusion_turb(
00576
       ctl_t * ctl,
       atm_t * atm,
00577
00578
       int ip,
00579
       double dt.
00580
       qsl_rnq * rnq) {
00581
00582
       double dx, dz, pt, p0, p1, w;
00583
00584
       /* Get tropopause pressure... */
00585
       pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00586
00587
        /* Get weighting factor... */
       p1 = pt * 0.866877899;
p0 = pt / 0.866877899;
00588
00589
00590
       if (atm->p[ip] > p0)
       w = 1;
else if (atm->p[ip] < p1)</pre>
00591
00592
00593
00594
       else
00595
         w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00596
00597
       /* Set diffusivity... */
dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00598
       dz = w * ctl - turb_dz_trop + (1 - w) * ctl - turb_dz_strat;
00599
00600
00601
        /* Horizontal turbulent diffusion... */
00602
       if (dx > 0) {
         atm->lon[ip]
00603
00604
           += DX2DEG(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 \star dx \star fabs(dt)))
00605
                     / 1000., atm->lat[ip]);
         atm->lat[ip]
00606
00607
           += DY2DEG(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00608
                     / 1000.);
00609
00610
00611
       /* Vertical turbulent diffusion... */
00612
       if (dz > 0)
00613
         atm->p[ip]
00614
           += DZ2DP(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dz * fabs(dt)))
00615
                    / 1000., atm->p[ip]);
00616 }
00617
00619
00620 void module_isosurf(
00621
       ctl_t * ctl,
       met_t * met0,
00622
00623
       met_t * met1,
       atm_t * atm,
00624
00625
       int ip) {
00626
00627
       static double *iso, *ps, t, *ts;
00628
00629
       static int idx, ip2, n;
00630
00631
       FILE *in;
00632
00633
       char line[LEN];
00634
00635
       /* Initialize... */
00636
       if (ip < 0) {</pre>
00637
          /* Allocate... */
00638
00639
         ALLOC(iso, double,
00640
              NP);
         ALLOC(ps, double,
00641
00642
               NP);
```

```
ALLOC(ts, double,
00644
                 NP);
00645
00646
           /* Save pressure... */
           if (ctl->isosurf == 1)
  for (ip2 = 0; ip2 < atm->np; ip2++)
00647
00648
               iso[ip2] = atm->p[ip2];
00649
00650
           /* Save density... */
00651
           else if (ctl->isosurf == 2)
00652
             for (ip2 = 0; ip2 < atm->np; ip2++) {
00653
               intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00654
                                 atm->lon[ip2], atm->lat[ip2], NULL, NULL, NULL,
00655
               iso[ip2] = atm->p[ip2] / t;
00656
00657
00658
00659
           /* Save potential temperature... */
else if (ctl->isosurf == 3)
00660
00661
             for (ip2 = 0; ip2 < atm->np; ip2++) {
00662
00663
               intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
                                 atm->lon[ip2], atm->lat[ip2], NULL, NULL, NULL,
00664
                                 &t, NULL, NULL, NULL, NULL, NULL, NULL);
00665
00666
               iso[ip2] = THETA(atm->p[ip2], t);
00667
00668
00669
           /* Read balloon pressure data... */
00670
           else if (ctl->isosurf == 4) {
00671
00672
             /* Write info... */
00673
             printf("Read balloon pressure data: %s\n", ctl->balloon);
00674
00675
00676
             if (!(in = fopen(ctl->balloon, "r")))
               ERRMSG("Cannot open file!");
00677
00678
00679
             /* Read pressure time series... */
             while (fgets(line, LEN, in))

if (sscanf(line, "%lg %lg", &ts[n], &ps[n]) == 2)
00680
00681
00682
                 if ((++n) > NP)
                    ERRMSG("Too many data points!");
00683
00684
00685
             /* Check number of points... */
00686
             if (n < 1)
               ERRMSG("Could not read any data!");
00687
00688
00689
             /* Close file... */
00690
             fclose(in);
           }
00691
00692
00693
           /* Leave initialization... */
00694
00695
00696
         /* Restore pressure... */
if (ctl->isosurf == 1)
00697
00698
          atm->p[ip] = iso[ip];
00700
        /* Restore density... */
else if (ctl->isosurf == 2) {
00701
00702
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00703
      lon[ip],
00704
                             atm->lat[ip], NULL, NULL, NULL, &t,
00705
                             NULL, NULL, NULL, NULL, NULL, NULL);
00706
           atm->p[ip] = iso[ip] * t;
00707
00708
        /* Restore potential temperature... */
else if (ctl->isosurf == 3) {
00709
00710
00711
           intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00712
                             atm->lat[ip], NULL, NULL, NULL, &t,
           NULL, NULL, NULL, NULL, NULL, NULL); atm->p[ip] = 1000. * pow(iso[ip] / t, -1. / 0.286);
00713
00714
00715
00716
00717
         /* Interpolate pressure... */
00718
         else if (ctl->isosurf == 4) {
          if (atm->time[ip] <= ts[0])</pre>
00719
           atm->p[ip] = ps[0];
else if (atm->time[ip] >= ts[n - 1])
00720
00721
00722
            atm->p[ip] = ps[n-1];
00723
             idx = locate_irr(ts, n, atm->time[ip]);
00724
             atm->p[ip] = LIN(ts[idx], ps[idx],
ts[idx + 1], ps[idx + 1], atm->time[ip]);
00725
00726
00727
           }
```

```
00728
00729 }
00730
00732
00733 void module_meteo(
00734
       ctl_t * ctl,
00735
        met_t * met0,
       met_t * met1,
atm_t * atm,
00736
00737
00738
       int ip) {
00739
00740
       double a, b, c, ps, pt, pv, p_hno3, p_h2o, t, u, v, w, x1, x2, h2o, o3, z;
00741
00742
        /* Interpolate meteorological data... */
00743
       intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
     lon[ip],
00744
                        atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &pv, &h2o, &o3);
00745
00746
        /* Set surface pressure... */
00747
       if (ctl->qnt_ps >= 0)
00748
          atm->q[ctl->qnt_ps][ip] = ps;
00749
00750
       /* Set tropopause pressure... */
if (ctl->gnt_pt >= 0)
00751
00752
         atm->q[ctl->qnt_pt][ip] = pt;
00753
00754
        /* Set pressure...
       if (ctl->qnt_p >= 0)
  atm->q[ctl->qnt_p][ip] = atm->p[ip];
00755
00756
00757
00758
        /* Set geopotential height... */
00759
       if (ctl->qnt_z >= 0)
00760
          atm->q[ctl->qnt_z][ip] = z;
00761
00762
        /* Set temperature... */
00763
        if (ctl->qnt t>= 0)
00764
          atm->q[ctl->qnt_t][ip] = t;
00765
00766
        /* Set zonal wind... */
00767
       if (ctl->qnt_u >= 0)
         atm->q[ctl->qnt_u][ip] = u;
00768
00769
00770
        /* Set meridional wind... */
00771
        if (ctl->qnt_v >= 0)
00772
          atm \rightarrow q[ctl \rightarrow qnt_v][ip] = v;
00773
00774
        /* Set vertical velocity... */
00775
        if (ctl->qnt_w >= 0)
         atm->q[ctl->qnt_w][ip] = w;
00776
00777
00778
        /* Set water vapor vmr... */
00779
        if (ct1->qnt_h2o >= 0)
00780
         atm->q[ctl->qnt_h2o][ip] = h2o;
00781
00782
        /* Set ozone vmr...
00783
       if (ctl->qnt_o3 >= 0)
00784
          atm \rightarrow q[ctl \rightarrow qnt_o3][ip] = o3;
00785
00786
       /* Calculate horizontal wind... */
00787
       if (ctl->qnt_vh >= 0)
00788
         atm->q[ctl->qnt\_vh][ip] = sqrt(u * u + v * v);
00789
00790
        /\star Calculate vertical velocity... \star/
00791
        if (ctl->qnt_vz >= 0)
00792
         atm->q[ctl->qnt\_vz][ip] = -1e3 * H0 / atm->p[ip] * w;
00793
        /\star Calculate potential temperature... \star/
00794
00795
       if (ctl->qnt_theta >= 0)
00796
          atm->q[ctl->qnt_theta][ip] = THETA(atm->p[ip], t);
00797
00798
        /* Set potential vorticity... */
        if (ctl->qnt_pv >= 0)
00799
00800
          atm->q[ctl->qnt_pv][ip] = pv;
00801
00802
        /* Calculate T_ice (Marti and Mauersberger, 1993)... */
00803
        if (ctl->qnt_tice >= 0)
00804
          atm->q[ctl->qnt_tice][ip] =
00805
            -2663.5 /
00806
            (\log 10((ct1-psc h2o > 0 ? ct1-psc h2o : h2o) * atm-p[ip] * 100.) -
00807
             12.537);
00808
00809
        /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
00810
        if (ctl->qnt_tnat >= 0) {
00811
         if (ctl->psc\_hno3 > 0)
00812
           p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
00813
          else
```

```
p_hno3 = clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip])
          * 1e-9 * atm->p[ip] / 1.333224;
p_h2o = (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
00815
00816
           a = 0.009179 - 0.00088 * log10(p_h2o);
00817
          b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
00818
00819
           c = -11397.0 / a;
          x1 = (-b + \text{sqrt}(b * b - 4. * c)) / 2.;

x2 = (-b - \text{sqrt}(b * b - 4. * c)) / 2.;
00821
00822
           if (x1 > 0)
          atm->q[ctl->qnt_tnat][ip] = x1;
if (x2 > 0)
00823
00824
00825
            atm->q[ctl->qnt_tnat][ip] = x2;
00826
00827
00828
         /* Calculate T_STS (mean of T_ice and T_NAT)... \star/
        if (ctl->qnt_tsts >= 0) {
   if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)</pre>
00829
00830
            ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
00831
           atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
00832
                                                 + atm->q[ctl->qnt_tnat][ip]);
00833
00834
00835 }
00836
00838
00839 void module_position(
00840
        met_t * met0,
00841
        met_t * met1,
        atm_t * atm,
00842
00843
        int ip) {
00844
00845
        double ps;
00846
00847
        /\star Calculate modulo... \star/
        atm->lon[ip] = fmod(atm->lon[ip], 360);
atm->lat[ip] = fmod(atm->lat[ip], 360);
00848
00849
00850
00851
        /* Check latitude... */
00852
        while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
         if (atm->lat[ip] > 90) {
  atm->lat[ip] = 180 - atm->lat[ip];
  atm->lon[ip] += 180;
00853
00854
00855
00856
00857
          if (atm->lat[ip] < -90) {</pre>
            atm->lat[ip] = -180 - atm->lat[ip];
00858
00859
             atm->lon[ip] += 180;
00860
00861
00862
        /* Check longitude... */
while (atm->lon[ip] < -180)</pre>
00863
00864
00865
          atm->lon[ip] += 360;
00866
        while (atm->lon[ip] >= 180)
00867
          atm->lon[ip] -= 360;
00868
00869
        /* Get surface pressure... */
intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00870
00871
                          atm->lon[ip], atm->lat[ip], &ps, NULL, NULL, NULL,
00872
                          NULL, NULL, NULL, NULL, NULL, NULL);
00873
00874
        /* Check pressure... */
        if (atm->p[ip] > ps)
00875
        atm->p[ip] = ps;
else if (atm->p[ip] < met0->p[met0->np - 1])
00876
00877
00878
          atm -> p[ip] = met0 -> p[met0 -> np - 1];
00879 }
00880
00882
00883 void module_sedi(
        ctl_t * ctl,
met_t * met0,
00884
00885
00886
        met_t * met1,
        atm_t * atm,
00887
00888
        int ip,
        double dt) {
00889
00890
00891
        /\star Coefficients for Cunningham slip-flow correction (Kasten, 1968): \star/
00892
        const double A = 1.249, B = 0.42, C = 0.87;
00893
00894
        /\star Average mass of an air molecule [kg/molec]: \star/
        const double m = 4.8096e-26;
00895
00896
00897
        double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
00898
        /* Convert units... */
00899
00900
        p = 100 * atm -> p[ip];
```

```
r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
00902
        rho_p = atm->q[ctl->qnt_rho][ip];
00903
00904
        /* Get temperature... */
        intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00905
      lon[ip],
00906
                           atm->lat[ip], NULL, NULL, NULL, &T,
00907
                           NULL, NULL, NULL, NULL, NULL, NULL);
00908
00909
         /\star Density of dry air... \star/
00910
        rho = p / (RA * T);
00911
        /* Dynamic viscosity of air... */
eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
00912
00913
00914
        /* Thermal velocity of an air molecule... */ v = sqrt(8 * KB * T / (M_PI * m));
00915
00916
00917
00918
        /* Mean free path of an air molecule... */
00919
        lambda = 2 * eta / (rho * v);
00920
00921
         /* Knudsen number for air... */
00922
        K = lambda / r_p;
00923
00924
         /* Cunningham slip-flow correction... */
00925
        G = 1 + K * (A + B * exp(-C / K));
00926
        /* Sedimentation (fall) velocity... */ v_p = 2. * SQR(r_p) * (rho_p - rho) * GO / (9. * eta) * G;
00927
00928
00929
00930
        /* Calculate pressure change... */
atm->p[ip] += DZ2DP(v_p * dt / 1000., atm->p[ip]);
00931
00932 }
00933
00935
00936 void write output(
        const char *dirname,
00938
         ctl_t * ctl,
00939
         met_t * met0,
00940
         met_t * met1,
00941
        atm t * atm,
00942
        double t) {
00943
00944
        char filename[2 * LEN];
00945
00946
        double r:
00947
00948
        int year, mon, day, hour, min, sec;
00949
00950
        /* Get time... */
00951
         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
00952
        /* Write atmospheric data... */
if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
    sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.tab",
00953
00954
00955
                   dirname, ctl->atm_basename, year, mon, day, hour, min);
00957
          write_atm(filename, ctl, atm, t);
00958
00959
00960
         /* Write CSI data... */
         if (ctl->csi_basename[0] != '-') {
00961
00962
          sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
00963
           write_csi(filename, ctl, atm, t);
00964
00965
00966
         /* Write ensemble data... */
if (ctl->ens_basename[0] != '-') {
00967
         sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
00968
           write_ens(filename, ctl, atm, t);
00969
00970
00971
00972
         /* Write gridded data... */
if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
00973
          sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
dirname, ctl->grid_basename, year, mon, day, hour, min);
00974
00975
00976
           write_grid(filename, ctl, met0, met1, atm, t);
00977
00978
00979
         /* Write profile data... */
         /* white profile data... */
if (ctl->prof_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
00980
00982
           write_prof(filename, ctl, met0, met1, atm, t);
00983
00984
00985
         /* Write station data... */
00986
         if (ctl->stat_basename[0] != '-') {
```

5.37 wind.c File Reference

Create meteorological data files with synthetic wind fields.

Functions

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

5.37.1 Detailed Description

Create meteorological data files with synthetic wind fields.

Definition in file wind.c.

5.37.2 Function Documentation

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

Definition at line 188 of file wind.c.

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

Definition at line 41 of file wind.c.

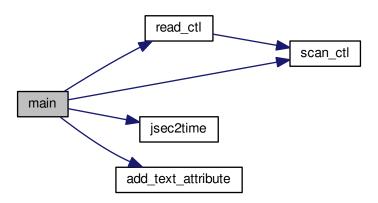
```
00043
00044
00045
        ctl_t ctl;
00046
00047
       static char filename[LEN];
00048
00049
       static double r, t0, z0, z1, dataLon[EX], dataLat[EY], dataZ[EP],
00050
          u0, u1, alpha;
00051
00052
       static float *dataT, *dataU, *dataV, *dataW;
00053
00054
       static int ncid, dims[4], timid, levid, latid, lonid, tid, uid, vid, wid,
00055
          idx, ix, iy, iz, nx, ny, nz, year, mon, day, hour, min, sec;
00056
        /* Allocate... */
00057
00058
       ALLOC(dataT, float,
              EP * EY * EX);
00059
00060
       ALLOC(dataU, float,
EP * EY * EX);
00061
00062
        ALLOC(dataV, float,
00063
              EP * EY * EX);
```

```
00064
              ALLOC (dataW, float,
00065
                         EP * EY * EX);
00066
00067
               /* Check arguments... */
00068
               if (argc < 3)
00069
                   ERRMSG("Give parameters: <ctl> <metbase>");
00070
00071
               /* Read control parameters... */
              /* Read control parameters... */
read_ctl(argv[1], argc, argv, &ctl);
t0 = scan_ctl(argv[1], argc, argv, "WIND_TO", -1, "0", NULL);
nx = (int) scan_ctl(argv[1], argc, argv, "WIND_NX", -1, "360", NULL);
ny = (int) scan_ctl(argv[1], argc, argv, "WIND_NY", -1, "181", NULL);
nz = (int) scan_ctl(argv[1], argc, argv, "WIND_NZ", -1, "61", NULL);
z0 = scan_ctl(argv[1], argc, argv, "WIND_ZO", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "WIND_Z1", -1, "60", NULL);
u0 = scan_ctl(argv[1], argc, argv, "WIND_UO", -1, "38.587660177302", NULL);
u1 = scan_ctl(argv[1], argc, argv, "WIND_U1", -1, "38.587660177302", NULL);
alpha = scan_ctl(argv[1], argc, argv, "WIND_L1PHA", -1, "0.0", NULL);
00072
00073
00074
00075
00076
00077
00078
00079
08000
               alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);
00081
00082
00083
                /* Check dimensions... */
              if (nx < 1 || nx > EX)
00084
00085
                  ERRMSG("Set 1 <= NX <= MAX!");
               if (ny < 1 || ny > EY)
   ERRMSG("Set 1 <= NY <= MAX!");</pre>
00086
00087
00088
               if (nz < 1 || nz > EP)
                  ERRMSG("Set 1 <= NZ <= MAX!");
00089
00090
00091
                /* Get time... */
00092
                jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
00093
               t0 = year * 10000. + mon * 100. + day + hour / 24.;
00094
00095
               /* Set filename... */
sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", argv[2], year, mon, day, hour);
00096
00097
                 /* Create netCDF file...
00098
               NC(nc_create(filename, NC_CLOBBER, &ncid));
00099
00100
                /* Create dimensions... *
00101
               /* Create dimensions... */
NC(nc_def_dim(ncid, "time", 1, &dims[0]));
NC(nc_def_dim(ncid, "lev", (size_t) nz, &dims[1]));
NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[2]));
NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[3]));
00102
00103
00104
00105
00106
00107
                /* Create variables... */
              /* Create variables... */
NC (nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));
NC (nc_def_var(ncid, "lev", NC_DOUBLE, 1, &dims[1], &levid));
NC (nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[2], &latid));
NC (nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[3], &lonid));
NC (nc_def_var(ncid, "T", NC_FLOAT, 4, &dims[0], &tid));
NC (nc_def_var(ncid, "U", NC_FLOAT, 4, &dims[0], &uid));
00109
00110
00111
00112
00113
               NC(nc_def_var(ncid, "V", NC_FLOAT, 4, &dims[0], &vid));
00114
00115
               NC(nc_def_var(ncid, "W", NC_FLOAT, 4, &dims[0], &wid));
00116
00117
              /* Set attributes... */
add_text_attribute(ncid, "time", "long_name", "time");
add_text_attribute(ncid, "time", "units", "day as %Y%m%d.%f");
add_text_attribute(ncid, "lon", "long_name", "longitude");
add_text_attribute(ncid, "lon", "units", "degrees_east");
add_text_attribute(ncid, "lat", "long_name", "latitude");
add_text_attribute(ncid, "lat", "units", "degrees_north");
add_text_attribute(ncid, "lev", "long_name", "air_pressure");
add_text_attribute(ncid, "lev", "units", "Pa");
add_text_attribute(ncid, "T", "long_name", "Temperature");
add_text_attribute(ncid, "U", "units", "K");
add_text_attribute(ncid, "U", "long_name", "U velocity");
add_text_attribute(ncid, "U", "units", "m s**-l");
                /* Set attributes... */
00118
00119
00121
00122
00123
00124
00125
00126
00127
00128
              add_text_attribute(ncid, "U", "long_name", "U Velocity");
add_text_attribute(ncid, "U", "units", "m s**-1");
add_text_attribute(ncid, "V", "long_name", "V velocity");
add_text_attribute(ncid, "V", "units", "m s**-1");
add_text_attribute(ncid, "W", "long_name", "Vertical velocity");
add_text_attribute(ncid, "W", "units", "Pa s**-1");
00129
00130
00131
00132
00133
00134
00135
               /* End definition... */
00136
              NC(nc_enddef(ncid));
00137
00138
                /* Set coordinates... */
               for (ix = 0; ix < nx; ix++)</pre>
00139
                  dataLon[ix] = 360.0 / nx * (double) ix;
00140
               for (iy = 0; iy < ny; iy++)
  dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;</pre>
00141
00142
               for (iz = 0; iz < nz; iz++)
00143
                  dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));
00144
00145
00146
                /* Write coordinates...
00147
               NC(nc_put_var_double(ncid, timid, &t0));
00148
               NC(nc_put_var_double(ncid, levid, dataZ));
00149
               NC(nc_put_var_double(ncid, lonid, dataLon));
00150
               NC(nc_put_var_double(ncid, latid, dataLat));
```

```
00152
          /* Create wind fields (Williamson et al., 1992)... */
00153
          for (ix = 0; ix < nx; ix++)
           for (iy = 0; iy < ny; iy++)</pre>
00154
              for (iz = 0; iz < nz; iz++) {
  idx = (iz * ny + iy) * nx + ix;</pre>
00155
00156
                 dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)

* (cos(dataLat[iy] * M_PI / 180.0)
00157
00158
00159
                                                * cos(alpha * M_PI / 180.0)
                                                + sin(dataLat[iy] * M_PI / 180.0)
* cos(dataLon[ix] * M_PI / 180.0)
00160
00161
                 * sin(alpha * M_PI / 180.0)));
dataV[idx] = (float) (-LIN(0.0, u0, nz - 1.0, u1, iz)
* sin(dataLon[ix] * M_PI / 180.0)
00162
00163
00164
00165
                                            * sin(alpha * M_PI / 180.0));
00166
00167
00168
          /* Write wind data... */
         NC(nc_put_var_float(ncid, tid, dataT));
00169
00170
         NC(nc_put_var_float(ncid, uid, dataU));
00171
         NC(nc_put_var_float(ncid, vid, dataV));
00172
         NC(nc_put_var_float(ncid, wid, dataW));
00173
00174
          /* Close file... */
00175
         NC(nc_close(ncid));
00176
00177
          /* Free... */
00178
         free(dataT);
00179
         free (dataU);
00180
         free (dataV):
00181
         free (dataW):
00182
00183
         return EXIT_SUCCESS;
00184 }
```

Here is the call graph for this function:



5.38 wind.c

```
00001 /*
           This file is part of MPTRAC.
00002
00003
           MPTRAC is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by
00004
00005
00006
           the Free Software Foundation, either version 3 of the License, or
00007
            (at your option) any later version.
80000
           MPTRAC is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00009
00010
00011
00012
           GNU General Public License for more details.
```

5.38 wind.c 237

```
00014
           You should have received a copy of the GNU General Public License
00015
           along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>.
00016
00017
           Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00025 #include "libtrac.h"
00026
00027 /* -----
00028
            Functions...
00029
00030
00031 void add_text_attribute(
00032
          int ncid,
00033
           char *varname,
00034
          char *attrname.
00035
          char *text);
00036
00037 /*
00038
00039
00040
00041 int main(
00042
           int argc,
00043
          char *argv[]) {
00044
00045
          ctl_t ctl;
00046
00047
          static char filename[LEN]:
00048
00049
          static double r, t0, z0, z1, dataLon[EX], dataLat[EY], dataZ[EP],
00050
             u0, u1, alpha;
00051
00052
          static float *dataT, *dataU, *dataV, *dataW;
00053
00054
          static int ncid, dims[4], timid, levid, latid, lonid, tid, uid, vid, wid,
00055
             idx, ix, iy, iz, nx, ny, nz, year, mon, day, hour, min, sec;
00056
00057
           /* Allocate... */
00058
          ALLOC(dataT, float,
00059
                   EP * EY * EX);
           ALLOC (dataU, float.
00060
00061
                    EP * EY * EX);
           ALLOC (dataV, float,
00062
00063
                   EP * EY * EX);
00064
          ALLOC (dataW, float,
00065
                   EP * EY * EX);
00066
00067
           /* Check arguments... */
00068
           if (argc < 3)
00069
             ERRMSG("Give parameters: <ctl> <metbase>");
00070
00071
           /* Read control parameters... */
          /* Read control parameters... */
read_ctl(argv[1], argc, argv, &ctl);
t0 = scan_ctl(argv[1], argc, argv, "WIND_TO", -1, "0", NULL);
nx = (int) scan_ctl(argv[1], argc, argv, "WIND_NX", -1, "360", NULL);
ny = (int) scan_ctl(argv[1], argc, argv, "WIND_NY", -1, "181", NULL);
nz = (int) scan_ctl(argv[1], argc, argv, "WIND_NZ", -1, "61", NULL);
z0 = scan_ctl(argv[1], argc, argv, "WIND_ZO", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "WIND_Z1", -1, "60", NULL);
u0 = scan_ctl(argv[1], argc, argv, "WIND_UO", -1, "38.587660177302", NULL);
u1 = scan_ctl(argv[1], argc, argv, "WIND_U1", -1, "38.587660177302", NULL);
alpha = scan_ctl(argv[1], argc, argv, "WIND_LD1PHA", -1, "0.0", NULL);
00072
00073
00075
00076
00077
00078
00079
08000
00081
           alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);
00082
00083
           /* Check dimensions... */
           if (nx < 1 || nx > EX)
   ERRMSG("Set 1 <= NX <= MAX!");</pre>
00084
00085
           if (ny < 1 || ny > EY)
ERRMSG("Set 1 <= NY <= MAX!");
00086
00087
88000
           if (nz < 1 || nz > EP)
00089
             ERRMSG("Set 1 <= NZ <= MAX!");</pre>
00090
00091
           /* Get time... */
           jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
t0 = year * 10000. + mon * 100. + day + hour / 24.;
00092
00093
00094
           /* Set filename... */
sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", argv[2], year, mon, day, hour);
00095
00096
00097
           /* Create netCDF file... */
00098
           NC(nc_create(filename, NC_CLOBBER, &ncid));
00099
00100
            /* Create dimensions... */
00101
           NC(nc_def_dim(ncid, "time", 1, &dims[0]));
NC(nc_def_dim(ncid, "lev", (size_t) nz, &dims[1]));
NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[2]));
00102
00103
00104
```

```
NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[3]));
00106
            /* Create variables... */
NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));
NC(nc_def_var(ncid, "lev", NC_DOUBLE, 1, &dims[1], &levid));
NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[2], &latid));
00107
00108
00109
00110
            NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[3], &lonid));
00111
            NC(nc_def_var(ncid, "T", NC_FLOAT, 4, &dims[0], &tid));
NC(nc_def_var(ncid, "U", NC_FLOAT, 4, &dims[0], &uid));
NC(nc_def_var(ncid, "V", NC_FLOAT, 4, &dims[0], &vid));
NC(nc_def_var(ncid, "W", NC_FLOAT, 4, &dims[0], &vid));
00112
00113
00114
00115
00116
00117
           /* Set attributes... */
add_text_attribute(ncid, "time", "long_name", "time");
add_text_attribute(ncid, "time", "units", "day as %Y%m%d.%f");
add_text_attribute(ncid, "lon", "long_name", "longitude");
add_text_attribute(ncid, "lon", "units", "degrees_east");
add_text_attribute(ncid, "lat", "long_name", "latitude");
add_text_attribute(ncid, "lat", "units", "degrees_north");
add_text_attribute(ncid, "lev", "long_name", "air_pressure");
add_text_attribute(ncid, "lev", "units", "Pa");
add_text_attribute(ncid, "T", "long_name", "Temperature");
add_text_attribute(ncid, "T", "units", "K");
add_text_attribute(ncid, "U", "long_name", "U velocity");
add_text_attribute(ncid, "U", "units", "m s**-1");
add_text_attribute(ncid, "V", "long_name", "V velocity");
            /* Set attributes... */
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
           add_text_attribute(ncid, "V", "long_name", "V velocity");
add_text_attribute(ncid, "V", "units", "m s**-1");
add_text_attribute(ncid, "W", "units", "m s**-1");
add_text_attribute(ncid, "W", "long_name", "Vertical velocity");
add_text_attribute(ncid, "W", "units", "Pa s**-1");
00130
00131
00132
00133
00134
00135
             /* End definition... */
00136
           NC(nc_enddef(ncid));
00137
00138
            /* Set coordinates... */
            for (ix = 0; ix < nx; ix++)
  dataLon[ix] = 360.0 / nx * (double) ix;</pre>
00139
00140
            for (iy = 0; iy < ny; iy++)
dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
00141
00143
            for (iz = 0; iz < nz; iz++)
00144
              dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));
00145
00146
            /* Write coordinates... */
00147
            NC(nc_put_var_double(ncid, timid, &t0));
00148
            NC(nc_put_var_double(ncid, levid, dataZ));
            NC(nc_put_var_double(ncid, lonid, dataLon));
00150
            NC(nc_put_var_double(ncid, latid, dataLat));
00151
00152
            /* Create wind fields (Williamson et al., 1992)... */
00153
            for (ix = 0; ix < nx; ix++)
              for (iy = 0; iy < ny; iy++)
00154
                  for (iz = 0; iz < nz; iz++) {
00155
00156
                      idx = (iz * ny + iy) * nx + ix;
00157
                      dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
                                                         * (cos(dataLat[iy] * M_PI / 180.0)

* cos(alpha * M_PI / 180.0)
00158
00159
                                                              + sin(dataLat[iy] * M_PI / 180.0)

* cos(dataLon[ix] * M_PI / 180.0)
00160
                                                              * sin(alpha * M_PI / 180.0)));
00162
                     00163
00164
                                                         * sin(alpha * M_PI / 180.0));
00165
00166
                  }
00167
00168
            /* Write wind data... */
00169
            NC(nc_put_var_float(ncid, tid, dataT));
00170
            NC(nc_put_var_float(ncid, uid, dataU));
00171
            NC(nc_put_var_float(ncid, vid, dataV));
00172
            NC(nc_put_var_float(ncid, wid, dataW));
00173
00174
             /* Close file...
00175
            NC(nc_close(ncid));
00176
00177
            /* Free... */
            free(dataT);
00178
00179
            free (dataU);
00180
            free (dataV):
00181
            free (dataW);
00182
00183
            return EXIT_SUCCESS;
00184 }
00185
00188 void add_text_attribute(
00189
           int ncid,
00190
            char *varname,
00191
           char *attrname.
```

5.38 wind.c 239

```
00192    char *text) {
00193
00194    int varid;
00195
00196    NC(nc_inq_varid(ncid, varname, &varid));
00197    NC(nc_put_att_text(ncid, varid, attrname, strlen(text), text));
00198 }
```

Index

add_text_attribute	libtrac.h, 139
wind.c, 234	csi_basename
atm_basename	ctl_t, 17
ctl_t, 16	csi dt out
atm conv.c, 25	ctl_t, 17
main, 26	csi_lat0
atm_dist.c, 27	ctl t, 18
main, 27	- :
atm_dt_out	csi_lat1
	ctl_t, 18
ctl_t, 17	csi_lon0
atm_filter	ctl_t, 18
ctl_t, 17	csi_lon1
atm_gpfile	ctl_t, 18
ctl_t, 16	csi_modmin
atm_init.c, 34	ctl_t, 17
main, 35	csi nx
atm_split.c, 38	ctl_t, 18
main, 38	csi_ny
atm_stat.c, 42	·
main, 42	ctl_t, 18
atm_t, 3	csi_nz
	ctl_t, 17
cache_time, 5	csi_obsfile
cache_usig, 5	ctl_t, 17
cache_vsig, 5	csi_obsmin
cache_wsig, 5	ctl_t, 17
lat, 5	csi_z0
lon, 4	_ ctl_t, 18
np, 4	csi z1
p, 4	-
q, 5	ctl_t, 18
time, 4	ctl_t, 6
up, 5	atm_basename, 16
vp, 5	atm_dt_out, 17
	atm_filter, 17
wp, 5	atm_gpfile, 16
atm_type	atm_type, 17
ctl_t, 17	balloon, 15
hallaaa	csi_basename, 17
balloon	csi_dt_out, 17
ctl_t, 15	csi_lat0, 18
anaha kima	csi lat1, 18
cache_time	csi_lon0, 18
atm_t, 5	
cache_usig	csi_lon1, 18
atm_t, 5	csi_modmin, 17
cache_vsig	csi_nx, 18
atm_t, 5	csi_ny, 18
cache_wsig	csi_nz, 17
atm_t, 5	csi_obsfile, 17
cart2geo	csi_obsmin, 17
libtrac.c, 55	csi_z0, 18
libtrac.h, 135	csi_z1, 18
clim hno3	direction, 13
libtrac.c, 55	dt_met, 13
libtrac.h, 135	
motracin, rec	Of Mod 13
clim trong	dt_mod, 13
clim_tropo libtrac.c, 58	ens_basename, 21 grid_basename, 18

grid_dt_out, 19	qnt_u, 12
grid_gpfile, 19	gnt unit, 10
grid_lat0, 20	qnt_v, 12
grid_lat1, 20	qnt vh, 12
grid_lon0, 19	qnt_vz, 12
-	• —
grid_lon1, 19	qnt_w, 12
grid_nx, 19	qnt_z, 11
grid_ny, 20	stat_basename, 21
grid_nz, 19	stat_lat, 21
grid_sparse, 19	stat_lon, 21
grid z0, 19	stat_r, <mark>22</mark>
grid_z1, 19	t_start, 13
isosurf, 15	t_stop, 13
	tdec_strat, 16
met_dp, 14	tdec_trop, 16
met_dt_out, 15	_ ·
met_dx, 14	turb_dx_strat, 15
met_dy, 14	turb_dx_trop, 15
met_geopot, 15	turb_dz_strat, 15
met_np, 14	turb_dz_trop, 15
met_p, 14	turb_mesox, 16
met sp, 14	turb_mesoz, 16
met_stage, 15	
	day2doy
met_sx, 14	libtrac.c, 60
met_sy, 14	libtrac.h, 141
met_tropo, 14	day2doy.c, 46
molmass, 16	•
nq, 10	main, 46
prof_basename, 20	direction
prof_lat0, 21	ctl_t, 13
prof_lat1, 21	doy2day
prof lon0, 21	libtrac.c, 60
prof_lon1, 21	libtrac.h, 141
prof_nx, 20	doy2day.c, 47
• —	main, 48
prof_ny, 21	dt met
prof_nz, 20	ctl_t, 13
prof_obsfile, 20	dt_mod
prof_z0, 20	ctl_t, 13
prof_z1, 20	σι <u>ι</u> , 10
psc_h2o, 16	
	ens hasename
psc_hno3, 16	ens_basename
psc_hno3, 16	_ ctl_t, 21
psc_hno3, 16 qnt_ens, 11	ctl_t, 21 extract.c, 49
psc_hno3, 16 qnt_ens, 11 qnt_format, 10	_ ctl_t, 21
psc_hno3, 16 qnt_ens, 11 qnt_format, 10 qnt_h2o, 12	ctl_t, 21 extract.c, 49 main, 49
psc_hno3, 16 qnt_ens, 11 qnt_format, 10 qnt_h2o, 12 qnt_m, 11	ctl_t, 21 extract.c, 49 main, 49 geo2cart
psc_hno3, 16 qnt_ens, 11 qnt_format, 10 qnt_h2o, 12 qnt_m, 11 qnt_name, 10	ctl_t, 21 extract.c, 49 main, 49 geo2cart libtrac.c, 61
psc_hno3, 16 qnt_ens, 11 qnt_format, 10 qnt_h2o, 12 qnt_m, 11 qnt_name, 10 qnt_o3, 12	ctl_t, 21 extract.c, 49 main, 49 geo2cart
psc_hno3, 16 qnt_ens, 11 qnt_format, 10 qnt_h2o, 12 qnt_m, 11 qnt_name, 10	ctl_t, 21 extract.c, 49 main, 49 geo2cart libtrac.c, 61
psc_hno3, 16 qnt_ens, 11 qnt_format, 10 qnt_h2o, 12 qnt_m, 11 qnt_name, 10 qnt_o3, 12	ctl_t, 21 extract.c, 49 main, 49 geo2cart libtrac.c, 61 libtrac.h, 141
psc_hno3, 16 qnt_ens, 11 qnt_format, 10 qnt_h2o, 12 qnt_m, 11 qnt_name, 10 qnt_o3, 12 qnt_p, 11	ctl_t, 21 extract.c, 49 main, 49 geo2cart libtrac.c, 61 libtrac.h, 141 get_met
psc_hno3, 16 qnt_ens, 11 qnt_format, 10 qnt_h2o, 12 qnt_m, 11 qnt_name, 10 qnt_o3, 12 qnt_p, 11 qnt_ps, 11	ctl_t, 21 extract.c, 49 main, 49 geo2cart libtrac.c, 61 libtrac.h, 141 get_met libtrac.c, 61
psc_hno3, 16 qnt_ens, 11 qnt_format, 10 qnt_h2o, 12 qnt_m, 11 qnt_name, 10 qnt_o3, 12 qnt_p, 11 qnt_ps, 11 qnt_pt, 11	ctl_t, 21 extract.c, 49 main, 49 geo2cart libtrac.c, 61 libtrac.h, 141 get_met libtrac.c, 61 libtrac.h, 142 get_met_help
psc_hno3, 16 qnt_ens, 11 qnt_format, 10 qnt_h2o, 12 qnt_m, 11 qnt_name, 10 qnt_o3, 12 qnt_p, 11 qnt_ps, 11 qnt_pt, 11 qnt_pt, 11 qnt_pv, 12 qnt_r, 11	ctl_t, 21 extract.c, 49 main, 49 geo2cart libtrac.c, 61 libtrac.h, 141 get_met libtrac.c, 61 libtrac.h, 142 get_met_help libtrac.c, 62
psc_hno3, 16 qnt_ens, 11 qnt_format, 10 qnt_h2o, 12 qnt_m, 11 qnt_name, 10 qnt_o3, 12 qnt_p, 11 qnt_ps, 11 qnt_pt, 11 qnt_pv, 12 qnt_r, 11 qnt_rho, 11	ctl_t, 21 extract.c, 49 main, 49 geo2cart libtrac.c, 61 libtrac.h, 141 get_met libtrac.h, 142 get_met_help libtrac.c, 62 libtrac.h, 143
psc_hno3, 16 qnt_ens, 11 qnt_format, 10 qnt_h2o, 12 qnt_m, 11 qnt_name, 10 qnt_o3, 12 qnt_p, 11 qnt_ps, 11 qnt_ps, 11 qnt_pt, 11 qnt_pv, 12 qnt_r, 11 qnt_rho, 11 qnt_stat, 13	ctl_t, 21 extract.c, 49 main, 49 geo2cart libtrac.c, 61 libtrac.h, 141 get_met libtrac.h, 142 get_met_help libtrac.c, 62 libtrac.h, 143 grid_basename
psc_hno3, 16 qnt_ens, 11 qnt_format, 10 qnt_h2o, 12 qnt_m, 11 qnt_name, 10 qnt_o3, 12 qnt_p, 11 qnt_ps, 11 qnt_ps, 11 qnt_pt, 11 qnt_pt, 11 qnt_rho, 11 qnt_stat, 13 qnt_t, 11	ctl_t, 21 extract.c, 49 main, 49 geo2cart libtrac.c, 61 libtrac.h, 141 get_met libtrac.h, 142 get_met_help libtrac.c, 62 libtrac.h, 143 grid_basename ctl_t, 18
psc_hno3, 16 qnt_ens, 11 qnt_format, 10 qnt_h2o, 12 qnt_m, 11 qnt_name, 10 qnt_o3, 12 qnt_p, 11 qnt_ps, 11 qnt_ps, 11 qnt_pt, 11 qnt_pt, 11 qnt_rho, 11 qnt_rho, 11 qnt_stat, 13 qnt_t, 11 qnt_theta, 12	ctl_t, 21 extract.c, 49 main, 49 geo2cart libtrac.c, 61 libtrac.h, 141 get_met libtrac.c, 61 libtrac.h, 142 get_met_help libtrac.c, 62 libtrac.h, 143 grid_basename ctl_t, 18 grid_dt_out
psc_hno3, 16 qnt_ens, 11 qnt_format, 10 qnt_h2o, 12 qnt_m, 11 qnt_name, 10 qnt_o3, 12 qnt_p, 11 qnt_ps, 11 qnt_pt, 11 qnt_pt, 11 qnt_rho, 11 qnt_rho, 11 qnt_stat, 13 qnt_t, 11 qnt_theta, 12 qnt_tice, 13	ctl_t, 21 extract.c, 49 main, 49 geo2cart libtrac.c, 61 libtrac.h, 141 get_met libtrac.c, 61 libtrac.h, 142 get_met_help libtrac.c, 62 libtrac.h, 143 grid_basename ctl_t, 18 grid_dt_out ctl_t, 19
psc_hno3, 16 qnt_ens, 11 qnt_format, 10 qnt_h2o, 12 qnt_m, 11 qnt_name, 10 qnt_o3, 12 qnt_p, 11 qnt_ps, 11 qnt_pt, 11 qnt_pt, 11 qnt_r, 11 qnt_rho, 11 qnt_rho, 11 qnt_stat, 13 qnt_t, 11 qnt_theta, 12 qnt_tice, 13 qnt_tnat, 13	ctl_t, 21 extract.c, 49 main, 49 geo2cart libtrac.c, 61 libtrac.h, 141 get_met libtrac.c, 61 libtrac.c, 61 libtrac.c, 62 libtrac.c, 62 libtrac.h, 143 grid_basename ctl_t, 18 grid_dt_out ctl_t, 19 grid_gpfile
psc_hno3, 16 qnt_ens, 11 qnt_format, 10 qnt_h2o, 12 qnt_m, 11 qnt_name, 10 qnt_o3, 12 qnt_p, 11 qnt_ps, 11 qnt_pt, 11 qnt_pt, 11 qnt_rho, 11 qnt_rho, 11 qnt_stat, 13 qnt_t, 11 qnt_theta, 12 qnt_tice, 13	ctl_t, 21 extract.c, 49 main, 49 geo2cart libtrac.c, 61 libtrac.h, 141 get_met libtrac.c, 61 libtrac.h, 142 get_met_help libtrac.c, 62 libtrac.h, 143 grid_basename ctl_t, 18 grid_dt_out ctl_t, 19

grid_lat0	intpol_met_space, 63
ctl_t, 20	intpol_met_time, 64
grid_lat1	jsec2time, 66
ctl_t, 20	locate_irr, 66
grid_lon0	locate_reg, 67
ctl_t, 19	read_atm, 67
grid_lon1	read_ctl, 69
ctl_t, 19	read_met, 73
grid_nx	read_met_extrapolate, 75
ctl_t, 19	read_met_geopot, 76
grid_ny	read_met_help, 78
ctl_t, 20	read_met_ml2pl, 78
grid_nz	read_met_periodic, 79
ctl_t, 19	read_met_pv, 79
grid_sparse	read_met_sample, 81
ctl_t, 19	read_met_tropo, 82
grid_z0	scan_ctl, 84
ctl_t, 19	time2jsec, 85
grid_z1	timer, 86
ctl_t, 19	write_atm, 86
h2o	write_csi, 88
	write_ens, 90
met_t, 25	write_grid, 92
intpol_met_2d	write_prof, 94
libtrac.c, 63	write_station, 96
libtrac.h, 143	libtrac.h, 133
intpol_met_3d	cart2geo, 135
libtrac.c, 63	clim_hno3, 135
libtrac.h, 144	clim_tropo, 139
intpol_met_space	day2doy, 141
libtrac.c, 63	doy2day, 141
libtrac.h, 144	geo2cart, 141
intpol_met_time	get_met, 142
libtrac.c, 64	get_met_help, 143
libtrac.h, 145	intpol_met_2d, 143
isosurf	intpol_met_3d, 144
ctl_t, 15	intpol_met_space, 144
	intpol_met_time, 145
jsec2time	jsec2time, 146
libtrac.c, 66	locate_irr, 147
libtrac.h, 146	locate_reg, 147 read_atm, 147
jsec2time.c, 51	read_ctl, 150
main, 52	read_cti, 150
lat	read_met_extrapolate, 155
lat	read_met_geopot, 156
atm_t, 5	read_met_help, 158
met_t, 23 libtrac.c, 53	read_met_ml2pl, 158
cart2geo, 55	read_met_periodic, 159
clim_hno3, 55	read_met_pv, 160
clim tropo, 58	read_met_sample, 161
day2doy, 60	read_met_tropo, 162
doy2day, 60	scan_ctl, 164
geo2cart, 61	time2jsec, 165
get_met, 61	timer, 166
get_met, 61 get_met_help, 62	write_atm, 166
intpol_met_2d, 63	write csi, 168
intpol_met_3d, 63	write_csi, 100 write ens, 170
poiou, 00	WING_0113, 170

write_grid, 172	met_t, 22
write_prof, 174	h2o, <mark>25</mark>
write_station, 176	lat, 23
locate_irr	lon, 23
libtrac.c, 66	np, <mark>23</mark>
libtrac.h, 147	nx, <mark>23</mark>
locate_reg	ny, <mark>23</mark>
libtrac.c, 67	o3, <mark>25</mark>
libtrac.h, 147	p, 24
lon	pl, 25
atm_t, 4	ps, 24
met_t, 23	pt, 24
main	pv, 24
atm_conv.c, 26	t, 24
atm_dist.c, 27	time, 23
atm_init.c, 35	u, 24
atm_split.c, 38	v, 24
atm stat.c, 42	w, 24
day2doy.c, 46	z, 24
doy2day.c, 48	met_tropo
extract.c, 49	ctl_t, 14 met_zm.c, 198
jsec2time.c, 52	main, 199
met_map.c, 186	module_advection
met_prof.c, 191	trac.c, 208
met_sample.c, 195	module_decay
met_zm.c, 199	trac.c, 208
smago.c, 202	module_diffusion_meso
time2jsec.c, 206	trac.c, 209
trac.c, 218	module_diffusion_turb
wind.c, 234	trac.c, 211
met_dp	module_isosurf
ctl_t, 14	trac.c, 212
met_dt_out	module_meteo
ctl_t, 15	trac.c, 213
met_dx	module_position
ctl_t, 14	trac.c, 215
met_dy	module_sedi
ctl_t, 14	trac.c, 216
met_geopot	molmass
ctl_t, 15	ctl_t, 16
met_map.c, 186	
main, 186	np
met_np ctl_t, 14	atm_t, 4
met p	met_t, 23
ctl_t, 14	nq
met_prof.c, 190	ctl_t, 10
main, 191	nx
met_sample.c, 194	met_t, 23
main, 195	ny mat t 00
met_sp	met_t, 23
ctl_t, 14	o3
met_stage	met t, 25
ctl_t, 15	
met_sx	р
ctl_t, 14	atm_t, 4
met_sy	met_t, 24
ctl_t, 14	pl

met_t, 25	ctl_t, 11
prof_basename	qnt_stat
ctl_t, 20	ctl_t, 13
prof_lat0	qnt_t
ctl_t, 21	ctl_t, 11
prof_lat1	qnt_theta
ctl_t, 21	ctl_t, 12
prof_lon0	qnt_tice
ctl_t, 21	ctl_t, 13
prof_lon1	qnt_tnat
ctl_t, 21	ctl_t, 13
prof_nx	qnt_tsts
ctl_t, 20	ctl_t, 13
prof_ny	qnt_u
ctl_t, 21	ctl_t, 12
prof_nz	qnt_unit
ctl t, 20	ctl_t, 10
prof_obsfile	qnt_v
ctl t, 20	ctl_t, 12
prof_z0	qnt_vh
ctl_t, 20	• —
	ctl_t, 12
prof_z1	qnt_vz
ctl_t, 20	ctl_t, 12
ps	qnt_w
met_t, 24	ctl_t, 12
psc_h2o	qnt_z
ctl_t, 16	ctl_t, 11
psc_hno3	road atm
ctl_t, 16	read_atm
pt	libtrac.c, 67
met_t, 24	libtrac.h, 147
met_t, 24 pv	read_ctl
	read_ctl libtrac.c, 69
pv met_t, 24	read_ctl libtrac.c, 69 libtrac.h, 150
pv met_t, 24 q	read_ctl libtrac.c, 69 libtrac.h, 150 read_met
pv met_t, 24	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73
pv met_t, 24 q atm_t, 5 qnt_ens	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11 qnt_format	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11 qnt_format ctl_t, 10	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11 qnt_format	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155 read_met_geopot
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11 qnt_format ctl_t, 10	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11 qnt_format ctl_t, 10 qnt_h2o ctl_t, 12 qnt_m	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155 read_met_geopot
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11 qnt_format ctl_t, 10 qnt_h2o ctl_t, 12	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155 read_met_geopot libtrac.c, 76
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11 qnt_format ctl_t, 10 qnt_h2o ctl_t, 12 qnt_m	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155 read_met_geopot libtrac.c, 76 libtrac.h, 156
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11 qnt_format ctl_t, 10 qnt_h2o ctl_t, 12 qnt_m ctl_t, 11	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155 read_met_geopot libtrac.c, 76 libtrac.h, 156 read_met_help
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11 qnt_format ctl_t, 10 qnt_h2o ctl_t, 12 qnt_m ctl_t, 11 qnt_name	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155 read_met_geopot libtrac.c, 76 libtrac.h, 156 read_met_help libtrac.c, 78
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11 qnt_format ctl_t, 10 qnt_h2o ctl_t, 12 qnt_m ctl_t, 11 qnt_name ctl_t, 10	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155 read_met_geopot libtrac.c, 76 libtrac.h, 156 read_met_help libtrac.c, 78 libtrac.h, 158
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11 qnt_format ctl_t, 10 qnt_h2o ctl_t, 12 qnt_m ctl_t, 11 qnt_name ctl_t, 10 qnt_name ctl_t, 10 qnt_o3	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155 read_met_geopot libtrac.c, 76 libtrac.h, 156 read_met_help libtrac.c, 78 libtrac.h, 158 read_met_ml2pl libtrac.c, 78
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11 qnt_format ctl_t, 10 qnt_h2o ctl_t, 12 qnt_m ctl_t, 11 qnt_name ctl_t, 10 qnt_o3 ctl_t, 12 qnt_p	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155 read_met_geopot libtrac.c, 76 libtrac.h, 156 read_met_help libtrac.c, 78 libtrac.h, 158 read_met_ml2pl libtrac.c, 78 libtrac.h, 158
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11 qnt_format ctl_t, 10 qnt_h2o ctl_t, 12 qnt_m ctl_t, 11 qnt_name ctl_t, 10 qnt_o3 ctl_t, 12	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155 read_met_geopot libtrac.c, 76 libtrac.h, 156 read_met_help libtrac.c, 78 libtrac.h, 158 read_met_ml2pl libtrac.c, 78
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11 qnt_format ctl_t, 10 qnt_h2o ctl_t, 12 qnt_m ctl_t, 11 qnt_name ctl_t, 10 qnt_o3 ctl_t, 12 qnt_p ctl_t, 11 qnt_ps	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155 read_met_geopot libtrac.c, 76 libtrac.h, 156 read_met_help libtrac.c, 78 libtrac.h, 158 read_met_ml2pl libtrac.c, 78 libtrac.h, 158 read_met_periodic libtrac.c, 79
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11 qnt_format ctl_t, 10 qnt_h2o ctl_t, 12 qnt_m ctl_t, 11 qnt_name ctl_t, 10 qnt_o3 ctl_t, 12 qnt_p ctl_t, 11 qnt_ps ctl_t, 11	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155 read_met_geopot libtrac.c, 76 libtrac.h, 156 read_met_help libtrac.c, 78 libtrac.h, 158 read_met_ml2pl libtrac.c, 78 libtrac.h, 158 read_met_periodic libtrac.c, 79 libtrac.h, 158
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11 qnt_format ctl_t, 10 qnt_h2o ctl_t, 12 qnt_m ctl_t, 11 qnt_name ctl_t, 10 qnt_o3 ctl_t, 12 qnt_p ctl_t, 11 qnt_ps ctl_t, 11 qnt_pt	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155 read_met_geopot libtrac.c, 76 libtrac.h, 156 read_met_help libtrac.c, 78 libtrac.h, 158 read_met_ml2pl libtrac.c, 78 libtrac.h, 158 read_met_periodic libtrac.c, 79 libtrac.c, 79 libtrac.h, 159 read_met_pv
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11 qnt_format ctl_t, 10 qnt_h2o ctl_t, 12 qnt_m ctl_t, 11 qnt_name ctl_t, 10 qnt_o3 ctl_t, 12 qnt_p ctl_t, 11 qnt_ps ctl_t, 11 qnt_pt ctl_t, 11	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155 read_met_geopot libtrac.c, 76 libtrac.h, 156 read_met_help libtrac.c, 78 libtrac.h, 158 read_met_ml2pl libtrac.c, 78 libtrac.h, 158 read_met_periodic libtrac.c, 79 libtrac.h, 159 read_met_pv libtrac.c, 79
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11 qnt_format ctl_t, 10 qnt_h2o ctl_t, 12 qnt_m ctl_t, 11 qnt_name ctl_t, 10 qnt_o3 ctl_t, 12 qnt_p ctl_t, 11 qnt_ps ctl_t, 11 qnt_pt ctl_t, 11 qnt_pv	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155 read_met_geopot libtrac.c, 76 libtrac.h, 156 read_met_help libtrac.c, 78 libtrac.h, 158 read_met_ml2pl libtrac.c, 78 libtrac.h, 158 read_met_periodic libtrac.c, 79 libtrac.h, 159 read_met_pv libtrac.c, 79 libtrac.h, 160
pv met_t, 24 q atm_t, 5 qnt_ens ctl_t, 11 qnt_format ctl_t, 10 qnt_h2o ctl_t, 12 qnt_m ctl_t, 11 qnt_name ctl_t, 10 qnt_o3 ctl_t, 12 qnt_p ctl_t, 11 qnt_ps ctl_t, 11 qnt_pt ctl_t, 11 qnt_pv ctl_t, 11 qnt_pv ctl_t, 11	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155 read_met_geopot libtrac.c, 76 libtrac.h, 156 read_met_help libtrac.c, 78 libtrac.h, 158 read_met_ml2pl libtrac.c, 78 libtrac.h, 158 read_met_periodic libtrac.c, 79 libtrac.h, 159 read_met_pv libtrac.c, 79 libtrac.c, 79 libtrac.h, 160 read_met_sample
pv	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155 read_met_geopot libtrac.c, 76 libtrac.h, 156 read_met_help libtrac.c, 78 libtrac.h, 158 read_met_ml2pl libtrac.c, 78 libtrac.h, 158 read_met_periodic libtrac.c, 79 libtrac.h, 159 read_met_pv libtrac.c, 79 libtrac.h, 160 read_met_sample libtrac.c, 81
pv	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155 read_met_geopot libtrac.c, 76 libtrac.h, 156 read_met_help libtrac.c, 78 libtrac.h, 158 read_met_ml2pl libtrac.c, 78 libtrac.h, 158 read_met_periodic libtrac.c, 79 libtrac.h, 159 read_met_pv libtrac.c, 79 libtrac.h, 160 read_met_sample libtrac.c, 81 libtrac.h, 161
pv	read_ctl libtrac.c, 69 libtrac.h, 150 read_met libtrac.c, 73 libtrac.h, 153 read_met_extrapolate libtrac.c, 75 libtrac.h, 155 read_met_geopot libtrac.c, 76 libtrac.h, 156 read_met_help libtrac.c, 78 libtrac.h, 158 read_met_ml2pl libtrac.c, 78 libtrac.h, 158 read_met_periodic libtrac.c, 79 libtrac.h, 159 read_met_pv libtrac.c, 79 libtrac.h, 160 read_met_sample libtrac.c, 81

libtrac.c, 82	turb_mesoz
libtrac.h, 162	ctl_t, 16
scan_ctl	u
libtrac.c, 84	met_t, 24
libtrac.h, 164	up
smago.c, 202	atm_t, 5
main, 202	
stat_basename	V mot t 24
ctl_t, 21	met_t, 24
stat_lat	vp
ctl_t, 21	atm_t, 5
stat_lon	W
ctl_t, 21	met t, 24
stat_r	wind.c, 234
ctl_t, 22	add_text_attribute, 234
•	main, 234
t	wp
met_t, 24	atm_t, 5
t_start	write_atm
ctl_t, 13	libtrac.c, 86
t_stop	libtrac.h, 166
ctl_t, 13	write_csi
tdec_strat	libtrac.c, 88
ctl_t, 16	libtrac.h, 168
tdec_trop ctl_t, 16	write_ens
time	libtrac.c, 90
atm_t, 4	libtrac.h, 170
met_t, 23	write_grid
time2jsec	libtrac.c, 92
libtrac.c, 85	libtrac.h, 172
libtrac.h, 165	write_output
time2jsec.c, 206	trac.c, 217
main, 206	write_prof
timer	libtrac.c, 94
libtrac.c, 86	libtrac.h, 174
libtrac.h, 166	write_station
trac.c, 207	libtrac.c, 96
main, 218	libtrac.h, 176
module_advection, 208	
module_decay, 208	Z
module_diffusion_meso, 209	met_t, 24
module_diffusion_turb, 211	
module_isosurf, 212	
module_meteo, 213	
module position, 215	
module_sedi, 216	
write_output, 217	
turb_dx_strat	
 ctl_t, 15	
turb_dx_trop	
ctl_t, 15	
turb_dz_strat	
ctl_t, 15	
turb_dz_trop	
ctl_t, 15	
turb_mesox	
ctl_t, 16	