# **MPTRAC**

Generated by Doxygen 1.8.11

ii CONTENTS

## Contents

1	Main	n Page		2			
2	Data	Data Structure Index					
	2.1	Data S	tructures	. 2			
3	File	Index		2			
	3.1	File Lis	st	. 2			
4	Data Structure Documentation						
	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	. 5			
		4.2.1	Detailed Description	. 10			
		4.2.2	Field Documentation	. 10			
	4.3	met_t S	Struct Reference	. 21			
		4.3.1	Detailed Description	. 22			
		4.3.2	Field Documentation	. 22			
5	File	Docume	entation	25			
	5.1	center.	c File Reference	. 25			
		5.1.1	Detailed Description	. 25			
		5.1.2	Function Documentation	. 25			
	5.2	center.	C	. 27			
	5.3	cluster.	c File Reference	. 29			
		5.3.1	Detailed Description	. 29			
		5.3.2	Function Documentation	. 29			
	5.4	cluster.	.с	. 32			
	5.5	conv.c	File Reference	. 35			
		5.5.1	Detailed Description	. 35			
		5.5.2	Function Documentation	. 35			

5.6	conv.c	36
5.7	day2doy.c File Reference	36
	5.7.1 Detailed Description	36
	5.7.2 Function Documentation	37
5.8	day2doy.c	37
5.9	dist.c File Reference	38
	5.9.1 Detailed Description	38
	5.9.2 Function Documentation	38
5.10	dist.c	41
5.11	doy2day.c File Reference	45
	5.11.1 Detailed Description	45
	5.11.2 Function Documentation	45
5.12	doy2day.c	46
5.13	extract.c File Reference	46
	5.13.1 Detailed Description	46
	5.13.2 Function Documentation	47
5.14	extract.c	48
5.15	init.c File Reference	49
	5.15.1 Detailed Description	49
	5.15.2 Function Documentation	49
5.16	init.c	51
5.17	jsec2time.c File Reference	53
	5.17.1 Detailed Description	53
	5.17.2 Function Documentation	53
5.18	jsec2time.c	54
5.19	libtrac.c File Reference	54
	5.19.1 Detailed Description	56
	5.19.2 Function Documentation	56
5.20	libtrac.c	97
5.21	libtrac.h File Reference	133

	5.21.1 Detailed Description	135
	5.21.2 Function Documentation	135
5.22	libtrac.h	176
5.23	match.c File Reference	185
	5.23.1 Detailed Description	185
	5.23.2 Function Documentation	185
5.24	match.c	187
5.25	met_map.c File Reference	189
	5.25.1 Detailed Description	189
	5.25.2 Function Documentation	190
5.26	met_map.c	192
5.27	met_prof.c File Reference	193
	5.27.1 Detailed Description	193
	5.27.2 Function Documentation	194
5.28	met_prof.c	196
5.29	met_sample.c File Reference	197
	5.29.1 Detailed Description	198
	5.29.2 Function Documentation	198
5.30	met_sample.c	199
5.31	met_zm.c File Reference	201
	5.31.1 Detailed Description	201
	5.31.2 Function Documentation	202
5.32	met_zm.c	203
5.33	smago.c File Reference	205
	5.33.1 Detailed Description	205
	5.33.2 Function Documentation	205
5.34	smago.c	207
5.35	split.c File Reference	208
	5.35.1 Detailed Description	208
	5.35.2 Function Documentation	209
5.36	split.c	211
5.37	time2jsec.c File Reference	212
	5.37.1 Detailed Description	212
	5.37.2 Function Documentation	213
5.38	time2jsec.c	213
5.39	trac.c File Reference	214
	5.39.1 Detailed Description	214
	5.39.2 Function Documentation	215
5.40	trac.c	229
5.41	wind.c File Reference	241
	5.41.1 Detailed Description	241
	5.41.2 Function Documentation	241
5.42	wind.c	243

Index 247

## 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
atl t	
ctl_t Control parameters	F
Control parameters	
met_t	
Meteorological data	21

## 3 File Index

## 3.1 File List

Here is a list of all files with brief descriptions:

center.c	
Calculate center of mass of air parcels	25
cluster.c	
Clustering of trajectories	29
conv.c	
Convert file format of atmospheric data files	35
day2doy.c	
Convert date to day of year	36
dist.c	
Calculate transport deviations of trajectories	38
doy2day.c	
Convert day of year to date	45

extract.c Extract single trajectory from atmospheric data files	46
init.c Create atmospheric data file with initial air parcel positions	49
jsec2time.c Convert Julian seconds to date	53
libtrac.c MPTRAC library definitions	54
libtrac.h MPTRAC library declarations	133
match.c Calculate deviations between two trajectories	185
met_map.c Extract global map from meteorological data	189
met_prof.c Extract vertical profile from meteorological data	193
met_sample.c Sample meteorological data at given geolocations	197
met_zm.c Extract zonal mean from meteorological data	201
smago.c Estimate horizontal diffusivity based on Smagorinsky theory	205
split.c Split air parcels into a larger number of parcels	208
time2jsec.c Convert date to Julian seconds	212
trac.c Lagrangian particle dispersion model	214
wind.c Create meteorological data files with synthetic wind fields	241

## **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]

Quantitiy 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].

## 4.1.1 Detailed Description

Atmospheric data.

Definition at line 561 of file libtrac.h.

- 4.1.2 Field Documentation
- 4.1.2.1 int atm\_t::np

Number of air pacels.

Definition at line 564 of file libtrac.h.

4.1.2.2 double atm\_t::time[NP]

Time [s].

Definition at line 567 of file libtrac.h.

4.1.2.3 double atm\_t::p[NP]

Pressure [hPa].

Definition at line 570 of file libtrac.h.

4.1.2.4 double atm\_t::lon[NP] Longitude [deg]. Definition at line 573 of file libtrac.h. 4.1.2.5 double atm\_t::lat[NP] Latitude [deg]. Definition at line 576 of file libtrac.h. 4.1.2.6 double atm\_t::q[NQ][NP] Quantitiy data (for various, user-defined attributes). Definition at line 579 of file libtrac.h. 4.1.2.7 float atm\_t::up[NP] Zonal wind perturbation [m/s]. Definition at line 582 of file libtrac.h. 4.1.2.8 float atm\_t::vp[NP] Meridional wind perturbation [m/s]. Definition at line 585 of file libtrac.h. 4.1.2.9 float atm\_t::wp[NP] Vertical velocity perturbation [hPa/s]. Definition at line 588 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. • char met\_stage [LEN] Command to stage meteo data. · int isosurf Isosurface parameter (0=none, 1=pressure, 2=density, 3=theta, 4=balloon). · char balloon [LEN] Balloon position filename. double turb\_dx\_trop Horizontal turbulent diffusion coefficient (troposphere) [m^2/s]. double turb dx strat Horizontal turbulent diffusion coefficient (stratosphere)  $[m^2/s]$ . double turb\_dz\_trop Vertical turbulent diffusion coefficient (troposphere)  $[m^2/s]$ . double turb\_dz\_strat Vertical turbulent diffusion coefficient (stratosphere)  $[m^2/2]$ . · double turb\_mesox

Horizontal scaling factor for mesoscale wind fluctuations.

Vertical scaling factor for mesoscale wind fluctuations.

Generated by Doxygen

double turb\_mesoz

· 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/m^2]. • int csi nz Number of altitudes of gridded CSI data. · double csi\_z0 Lower altitude of gridded CSI data [km]. double csi z1 Upper altitude of gridded CSI data [km]. · int csi nx Number of longitudes of gridded CSI data. double csi\_lon0 Lower longitude of gridded CSI data [deg]. double csi lon1 Upper longitude of gridded CSI data [deg]. • int csi\_ny Number of latitudes of gridded CSI data. double csi lat0 Lower latitude of gridded CSI data [deg]. · double csi\_lat1 Upper latitude of gridded CSI data [deg]. char 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 245 of file libtrac.h.
4.2.2 Field Documentation
4.2.2.1 int ctl_t::nq
Number of quantities.
Definition at line 248 of file libtrac.h.
4.2.2.2 char ctl_t::qnt_name[NQ][LEN]
Quantity names.
Definition at line 251 of file libtrac.h.
4.2.2.3 char ctl_t::qnt_unit[NQ][LEN]
Quantity units.
Definition at line 254 of file libtrac.h.
4.2.2.4 char ctl_t::qnt_format[NQ][LEN]
Quantity output format.
Definition at line 257 of file libtrac.h.
4.2.2.5 int ctl_t::qnt_ens
Quantity array index for ensemble IDs.
Definition at line 260 of file libtrac.h.
4.2.2.6 int ctl_t::qnt_m
Quantity array index for mass.
Definition at line 263 of file libtrac.h.
```

4.2.2.7 int ctl\_t::qnt\_rho Quantity array index for particle density. Definition at line 266 of file libtrac.h. 4.2.2.8 int ctl\_t::qnt\_r Quantity array index for particle radius. Definition at line 269 of file libtrac.h. 4.2.2.9 int ctl\_t::qnt\_ps Quantity array index for surface pressure. Definition at line 272 of file libtrac.h. 4.2.2.10 int ctl\_t::qnt\_pt Quantity array index for tropopause pressure. Definition at line 275 of file libtrac.h. 4.2.2.11 int ctl\_t::qnt\_z Quantity array index for geopotential height. Definition at line 278 of file libtrac.h. 4.2.2.12 int ctl\_t::qnt\_p Quantity array index for pressure. Definition at line 281 of file libtrac.h. 4.2.2.13 int ctl\_t::qnt\_t Quantity array index for temperature. Definition at line 284 of file libtrac.h. 4.2.2.14 int ctl\_t::qnt\_u Quantity array index for zonal wind. Definition at line 287 of file libtrac.h. 4.2.2.15 int ctl\_t::qnt\_v

Generated by Doxygen

Quantity array index for meridional wind.

Definition at line 290 of file libtrac.h.

```
4.2.2.16 int ctl_t::qnt_w
Quantity array index for vertical velocity.
Definition at line 293 of file libtrac.h.
4.2.2.17 int ctl_t::qnt_h2o
Quantity array index for water vapor vmr.
Definition at line 296 of file libtrac.h.
4.2.2.18 int ctl_t::qnt_o3
Quantity array index for ozone vmr.
Definition at line 299 of file libtrac.h.
4.2.2.19 int ctl_t::qnt_theta
Quantity array index for potential temperature.
Definition at line 302 of file libtrac.h.
4.2.2.20 int ctl_t::qnt_vh
Quantity array index for horizontal wind.
Definition at line 305 of file libtrac.h.
4.2.2.21 int ctl_t::qnt_vz
Quantity array index for vertical velocity.
Definition at line 308 of file libtrac.h.
4.2.2.22 int ctl_t::qnt_pv
Quantity array index for potential vorticity.
Definition at line 311 of file libtrac.h.
4.2.2.23 int ctl_t::qnt_tice
Quantity array index for T_ice.
Definition at line 314 of file libtrac.h.
4.2.2.24 int ctl_t::qnt_tsts
Quantity array index for T_STS.
Definition at line 317 of file libtrac.h.
```

```
4.2.2.25 int ctl_t::qnt_tnat
Quantity array index for T_NAT.
Definition at line 320 of file libtrac.h.
4.2.2.26 int ctl_t::qnt_stat
Quantity array index for station flag.
Definition at line 323 of file libtrac.h.
4.2.2.27 int ctl_t::direction
Direction flag (1=forward calculation, -1=backward calculation).
Definition at line 326 of file libtrac.h.
4.2.2.28 double ctl_t::t_start
Start time of simulation [s].
Definition at line 329 of file libtrac.h.
4.2.2.29 double ctl_t::t_stop
Stop time of simulation [s].
Definition at line 332 of file libtrac.h.
4.2.2.30 double ctl_t::dt_mod
Time step of simulation [s].
Definition at line 335 of file libtrac.h.
4.2.2.31 double ctl_t::dt_met
Time step of meteorological data [s].
Definition at line 338 of file libtrac.h.
4.2.2.32 int ctl_t::met_dx
Stride for longitudes.
Definition at line 341 of file libtrac.h.
4.2.2.33 int ctl_t::met_dy
Stride for latitudes.
Definition at line 344 of file libtrac.h.
```

```
4.2.2.34 int ctl_t::met_dp
Stride for pressure levels.
Definition at line 347 of file libtrac.h.
4.2.2.35 int ctl_t::met_sx
Smoothing for longitudes.
Definition at line 350 of file libtrac.h.
4.2.2.36 int ctl_t::met_sy
Smoothing for latitudes.
Definition at line 353 of file libtrac.h.
4.2.2.37 int ctl_t::met_sp
Smoothing for pressure levels.
Definition at line 356 of file libtrac.h.
4.2.2.38 int ctl_t::met_np
Number of target pressure levels.
Definition at line 359 of file libtrac.h.
4.2.2.39 double ctl_t::met_p[EP]
Target pressure levels [hPa].
Definition at line 362 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 366 of file libtrac.h.
4.2.2.41 char ctl_t::met_geopot[LEN]
Surface geopotential data file.
Definition at line 369 of file libtrac.h.
4.2.2.42 char ctl_t::met_stage[LEN]
Command to stage meteo data.
Definition at line 372 of file libtrac.h.
```

```
4.2.2.43 int ctl_t::isosurf
Isosurface parameter (0=none, 1=pressure, 2=density, 3=theta, 4=balloon).
Definition at line 376 of file libtrac.h.
4.2.2.44 char ctl_t::balloon[LEN]
Balloon position filename.
Definition at line 379 of file libtrac.h.
4.2.2.45 double ctl_t::turb_dx_trop
Horizontal turbulent diffusion coefficient (troposphere) [m^2/s].
Definition at line 382 of file libtrac.h.
4.2.2.46 double ctl_t::turb_dx_strat
Horizontal turbulent diffusion coefficient (stratosphere) [m<sup>2</sup>/s].
Definition at line 385 of file libtrac.h.
4.2.2.47 double ctl_t::turb_dz_trop
Vertical turbulent diffusion coefficient (troposphere) [m<sup>2</sup>/s].
Definition at line 388 of file libtrac.h.
4.2.2.48 double ctl_t::turb_dz_strat
Vertical turbulent diffusion coefficient (stratosphere) [m<sup>2</sup>/s].
Definition at line 391 of file libtrac.h.
4.2.2.49 double ctl_t::turb_mesox
Horizontal scaling factor for mesoscale wind fluctuations.
Definition at line 394 of file libtrac.h.
4.2.2.50 double ctl_t::turb_mesoz
Vertical scaling factor for mesoscale wind fluctuations.
Definition at line 397 of file libtrac.h.
4.2.2.51 double ctl_t::molmass
Molar mass [g/mol].
Definition at line 400 of file libtrac.h.
```

```
4.2.2.52 double ctl_t::tdec_trop
Life time of particles (troposphere) [s].
Definition at line 403 of file libtrac.h.
4.2.2.53 double ctl_t::tdec_strat
Life time of particles (stratosphere) [s].
Definition at line 406 of file libtrac.h.
4.2.2.54 double ctl_t::psc_h2o
H2O volume mixing ratio for PSC analysis.
Definition at line 409 of file libtrac.h.
4.2.2.55 double ctl_t::psc_hno3
HNO3 volume mixing ratio for PSC analysis.
Definition at line 412 of file libtrac.h.
4.2.2.56 char ctl_t::atm_basename[LEN]
Basename of atmospheric data files.
Definition at line 415 of file libtrac.h.
4.2.2.57 char ctl_t::atm_gpfile[LEN]
Gnuplot file for atmospheric data.
Definition at line 418 of file libtrac.h.
4.2.2.58 double ctl_t::atm_dt_out
Time step for atmospheric data output [s].
Definition at line 421 of file libtrac.h.
4.2.2.59 int ctl_t::atm_filter
Time filter for atmospheric data output (0=no, 1=yes).
Definition at line 424 of file libtrac.h.
4.2.2.60 int ctl_t::atm_type
Type of atmospheric data files (0=ASCII, 1=binary, 2=netCDF).
Definition at line 427 of file libtrac.h.
```

4.2.2.61 char ctl\_t::csi\_basename[LEN] Basename of CSI data files. Definition at line 430 of file libtrac.h. 4.2.2.62 double ctl\_t::csi\_dt\_out Time step for CSI data output [s]. Definition at line 433 of file libtrac.h. 4.2.2.63 char ctl\_t::csi\_obsfile[LEN] Observation data file for CSI analysis. Definition at line 436 of file libtrac.h. 4.2.2.64 double ctl\_t::csi\_obsmin Minimum observation index to trigger detection. Definition at line 439 of file libtrac.h. 4.2.2.65 double ctl\_t::csi\_modmin Minimum column density to trigger detection [kg/m<sup>2</sup>]. Definition at line 442 of file libtrac.h. 4.2.2.66 int ctl\_t::csi\_nz Number of altitudes of gridded CSI data. Definition at line 445 of file libtrac.h. 4.2.2.67 double ctl\_t::csi\_z0 Lower altitude of gridded CSI data [km]. Definition at line 448 of file libtrac.h. 4.2.2.68 double ctl\_t::csi\_z1 Upper altitude of gridded CSI data [km]. Definition at line 451 of file libtrac.h. 4.2.2.69 int ctl\_t::csi\_nx Number of longitudes of gridded CSI data. Definition at line 454 of file libtrac.h.

```
4.2.2.70 double ctl_t::csi_lon0
Lower longitude of gridded CSI data [deg].
Definition at line 457 of file libtrac.h.
4.2.2.71 double ctl_t::csi_lon1
Upper longitude of gridded CSI data [deg].
Definition at line 460 of file libtrac.h.
4.2.2.72 int ctl_t::csi_ny
Number of latitudes of gridded CSI data.
Definition at line 463 of file libtrac.h.
4.2.2.73 double ctl_t::csi_lat0
Lower latitude of gridded CSI data [deg].
Definition at line 466 of file libtrac.h.
4.2.2.74 double ctl_t::csi_lat1
Upper latitude of gridded CSI data [deg].
Definition at line 469 of file libtrac.h.
4.2.2.75 char ctl_t::grid_basename[LEN]
Basename of grid data files.
Definition at line 472 of file libtrac.h.
4.2.2.76 char ctl_t::grid_gpfile[LEN]
Gnuplot file for gridded data.
Definition at line 475 of file libtrac.h.
4.2.2.77 double ctl_t::grid_dt_out
Time step for gridded data output [s].
Definition at line 478 of file libtrac.h.
4.2.2.78 int ctl_t::grid_sparse
Sparse output in grid data files (0=no, 1=yes).
Definition at line 481 of file libtrac.h.
```

4.2.2.79 int ctl\_t::grid\_nz Number of altitudes of gridded data. Definition at line 484 of file libtrac.h. 4.2.2.80 double ctl\_t::grid\_z0 Lower altitude of gridded data [km]. Definition at line 487 of file libtrac.h. 4.2.2.81 double ctl\_t::grid\_z1 Upper altitude of gridded data [km]. Definition at line 490 of file libtrac.h. 4.2.2.82 int ctl\_t::grid\_nx Number of longitudes of gridded data. Definition at line 493 of file libtrac.h. 4.2.2.83 double ctl\_t::grid\_lon0 Lower longitude of gridded data [deg]. Definition at line 496 of file libtrac.h. 4.2.2.84 double ctl\_t::grid\_lon1 Upper longitude of gridded data [deg]. Definition at line 499 of file libtrac.h. 4.2.2.85 int ctl\_t::grid\_ny Number of latitudes of gridded data. Definition at line 502 of file libtrac.h. 4.2.2.86 double ctl\_t::grid\_lat0 Lower latitude of gridded data [deg]. Definition at line 505 of file libtrac.h. 4.2.2.87 double ctl\_t::grid\_lat1 Upper latitude of gridded data [deg].

Definition at line 508 of file libtrac.h.

```
4.2.2.88 char ctl_t::prof_basename[LEN]
Basename for profile output file.
Definition at line 511 of file libtrac.h.
4.2.2.89 char ctl_t::prof_obsfile[LEN]
Observation data file for profile output.
Definition at line 514 of file libtrac.h.
4.2.2.90 int ctl_t::prof_nz
Number of altitudes of gridded profile data.
Definition at line 517 of file libtrac.h.
4.2.2.91 double ctl_t::prof_z0
Lower altitude of gridded profile data [km].
Definition at line 520 of file libtrac.h.
4.2.2.92 double ctl_t::prof_z1
Upper altitude of gridded profile data [km].
Definition at line 523 of file libtrac.h.
4.2.2.93 int ctl_t::prof_nx
Number of longitudes of gridded profile data.
Definition at line 526 of file libtrac.h.
4.2.2.94 double ctl_t::prof_lon0
Lower longitude of gridded profile data [deg].
Definition at line 529 of file libtrac.h.
4.2.2.95 double ctl_t::prof_lon1
Upper longitude of gridded profile data [deg].
Definition at line 532 of file libtrac.h.
4.2.2.96 int ctl_t::prof_ny
Number of latitudes of gridded profile data.
Definition at line 535 of file libtrac.h.
```

```
4.2.2.97 double ctl_t::prof_lat0
Lower latitude of gridded profile data [deg].
Definition at line 538 of file libtrac.h.
4.2.2.98 double ctl_t::prof_lat1
Upper latitude of gridded profile data [deg].
Definition at line 541 of file libtrac.h.
4.2.2.99 char ctl_t::ens_basename[LEN]
Basename of ensemble data file.
Definition at line 544 of file libtrac.h.
4.2.2.100 char ctl_t::stat_basename[LEN]
Basename of station data file.
Definition at line 547 of file libtrac.h.
4.2.2.101 double ctl_t::stat_lon
Longitude of station [deg].
Definition at line 550 of file libtrac.h.
4.2.2.102 double ctl_t::stat_lat
Latitude of station [deg].
Definition at line 553 of file libtrac.h.
4.2.2.103 double ctl_t::stat_r
Search radius around station [km].
Definition at line 556 of file libtrac.h.
The documentation for this struct was generated from the following file:
    · libtrac.h
4.3 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 593 of file libtrac.h.

## 4.3.2 Field Documentation

#### 4.3.2.1 double met\_t::time

Time [s].

Definition at line 596 of file libtrac.h.

4.3.2.2 int met\_t::nx Number of longitudes. Definition at line 599 of file libtrac.h. 4.3.2.3 int met\_t::ny Number of latitudes. Definition at line 602 of file libtrac.h. 4.3.2.4 int met\_t::np Number of pressure levels. Definition at line 605 of file libtrac.h. 4.3.2.5 double met\_t::lon[EX] Longitude [deg]. Definition at line 608 of file libtrac.h. 4.3.2.6 double met\_t::lat[EY] Latitude [deg]. Definition at line 611 of file libtrac.h. 4.3.2.7 double met\_t::p[EP] Pressure [hPa]. Definition at line 614 of file libtrac.h. 4.3.2.8 double met\_t::ps[EX][EY] Surface pressure [hPa]. Definition at line 617 of file libtrac.h. 4.3.2.9 double met\_t::pt[EX][EY] Tropopause pressure [hPa]. Definition at line 620 of file libtrac.h. 4.3.2.10 float met\_t::z[EX][EY][EP] Geopotential height [km]. Definition at line 623 of file libtrac.h.

```
4.3.2.11 float met_t::t[EX][EY][EP]
Temperature [K].
Definition at line 626 of file libtrac.h.
4.3.2.12 float met_t::u[EX][EY][EP]
Zonal wind [m/s].
Definition at line 629 of file libtrac.h.
4.3.2.13 float met_t::v[EX][EY][EP]
Meridional wind [m/s].
Definition at line 632 of file libtrac.h.
4.3.2.14 float met_t::w[EX][EY][EP]
Vertical wind [hPa/s].
Definition at line 635 of file libtrac.h.
4.3.2.15 float met_t::pv[EX][EY][EP]
Potential vorticity [PVU].
Definition at line 638 of file libtrac.h.
4.3.2.16 float met_t::h2o[EX][EY][EP]
Water vapor volume mixing ratio [1].
Definition at line 641 of file libtrac.h.
4.3.2.17 float met_t::o3[EX][EY][EP]
Ozone volume mixing ratio [1].
Definition at line 644 of file libtrac.h.
4.3.2.18 float met_t::pl[EX][EY][EP]
Pressure on model levels [hPa].
Definition at line 647 of file libtrac.h.
The documentation for this struct was generated from the following file:
```

· libtrac.h

Generated by Doxygen

5 File Documentation 25

## 5 File Documentation

#### 5.1 center.c File Reference

Calculate center of mass of air parcels.

#### **Functions**

• int main (int argc, char \*argv[])

## 5.1.1 Detailed Description

Calculate center of mass of air parcels.

Definition in file center.c.

#### 5.1.2 Function Documentation

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

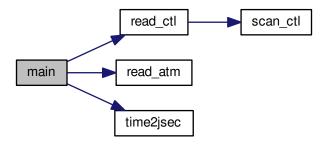
Definition at line 27 of file center.c.

```
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> <atm1> [<atm2> ...]");
00049
00050
        /* Read control parameters... */
00051
        read_ctl(argv[1], argc, argv, &ctl);
00052
00053
        /* Write info... */
00054
        printf("Write center of mass data: %s\n", argv[2]);
00055
        /* Create output file... */
if (!(out = fopen(argv[2], "w")))
00056
00057
         ERRMSG("Cannot create file!");
00058
00059
00060
        /* 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"
00066
                 "# $5 = altitude (10%% percentile) [km]\n"
00067
                 "# $6 = altitude (1st quarter) [km]\n"
                 "# $7 = altitude (median) [km]\n"
00068
                "# $8 = altitude (3rd quarter) [km]\n"
"# $9 = altitude (90%% percentile) [km]\n"
00069
00070
00071
                 "# $10 = altitude (maximum) [km] \n");
00072
       fprintf(out,
```

```
"# $11 = longitude (mean) [deg] n"
00074
                 "# $12 = longitude (sigma) [deg]\n"
                 "# $13 = longitude (minimum) [deg] \n"
00075
                 "# $14 = longitude (10%% percentile) [deg]\n"
00076
                 "# $15 = longitude (1st quarter) [deg] n
00077
00078
                 "# $16 = longitude (median) [deg]\n"
                 "# $17 = longitude (3rd quarter) [deg]\n"
00079
00080
                 "# $18 = longitude (90%% percentile)
00081
                 "# $19 = longitude (maximum) [deg]\n");
        fprintf(out, "# $20 = latitude (mean) [deg]\n
00082
00083
                 "# $21 = latitude (sigma) [deg]\n"
00084
                 "# $22 = latitude (minimum) [deg]\n"
00085
                 "# $23 = latitude (10%% percentile) [deg]\n"
00086
00087
                 "# $24 = latitude (1st quarter) [deg] \n"
                 "# $25 = latitude (median) [deg] n"
00088
                 "# $26 = latitude (3rd quarter) [deg]\n"
00089
                 "# $27 = latitude (90%% percentile) [deg]\n"
00090
                 "# $28 = latitude (maximum) [deg]\n\n");
00091
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
00102
          latm = lats = 0:
00103
00104
           /* Calculate mean and standard deviation... */
00105
          for (ip = 0; ip < atm->np; ip++) {
            zm += Z(atm->p[ip]) / atm->np;
00106
             lonm += atm->lon[ip] / atm->np;
latm += atm->lat[ip] / atm->np;
00107
00108
             zs += gsl_pow_2(Z(atm->p[ip])) / atm->np;
00109
             lons += gsl_pow_2(atm->lon[ip]) / atm->np;
00110
00111
             lats += gsl_pow_2(atm->lat[ip]) / atm->np;
00112
00113
          /* Normalize... */
00114
          zs = sqrt(zs - gsl_pow_2(zm));
00115
          lons = sqrt(lons - gsl_pow_2(lonm));
lats = sqrt(lats - gsl_pow_2(latm));
00116
00117
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
           /* Get time from filename... */
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
          /* Write data... */
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 / 2]), Z(atm->p[atm->np / 4]), Z(atm->p[atm->np / 10]), Z(atm->p[0]),
00143
00144
                   lonm, lons, atm->lon[0], atm->lon[atm->np / 10],
00145
                   atm->lon[atm->np / 4], atm->lon[atm->np / 2],
atm->lon[atm->np - atm->np / 4],
atm->lon[atm->np - atm->np / 10],
00146
00147
00148
00149
                   atm->lon[atm->np - 1],
                   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
00152
                   atm->lat[atm->np - atm->np / 10], atm->lat[atm->np - 1]);
00153
00154
00155
00156
        /* Close file... */
00157
        fclose(out);
00158
00159
        /* Free... */
```

5.2 center.c 27

Here is the call graph for this function:



#### 5.2 center.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.
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) 2013-2018 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028
        int argc,
00029
        char *argv[]) {
00030
00031
        ctl t ctl;
00032
00033
        atm_t *atm;
00034
00035
        FILE *out;
00036
00037
        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
          ERRMSG("Give parameters: <ctl> <outfile> <atml> [<atm2> ...]");
00048
00049
00050
        /* Read control parameters... */
00051
        read_ctl(argv[1], argc, argv, &ctl);
00052
```

```
/* 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
          ERRMSG("Cannot create file!");
00058
00060
         /* Write header... */
00061
        fprintf(out,
00062
                  "# $1
                        = 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"
00067
00068
                 "# $7 = altitude (median) [km]\n"
                 "# $8 = altitude (3rd quarter) [km] n"
00069
                 "# $9 = altitude (90%% percentile) [km]\n"
00070
                 "# $10 = altitude (maximum) [km] n");
00072
        fprintf(out,
                 "# $11 = longitude (mean) [deg]\n"
00073
                 "# $12 = longitude (sigma) [deg]\n"
00074
00075
                 "# $13 = longitude (minimum) [deg] \n"
                 "# $14 = longitude (10%% percentile) [deg]\n"
00076
00077
                 "# $15 = longitude (1st quarter) [deg]\n'
                 "# $16 = longitude (median) [deg]\n"
00078
00079
                 "# $17 = longitude (3rd quarter) [deg]\n"
08000
                 "# $18 = longitude (90%% percentile) [deg]\n"
                 "# $19 = longitude (maximum) [deg]\n");
00081
00082
        fprintf(out,
00083
                  "# $20 = latitude (mean) [deg] n
00084
                 "# $21 = latitude (sigma) [deg]\n"
00085
                 "# $22 = latitude (minimum) [deg] \n"
00086
                 "# $23 = latitude (10%% percentile) [deg] n"
                 "# $24 = latitude (1st quarter) [deg] \n"
00087
                 "# $25 = latitude (median) [deg]\n"
00088
                 "# $26 = latitude (3rd quarter) [deg]\n"
"# $27 = latitude (90%% percentile) [deg]\n"
00089
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
           /\star Calculate mean and standard deviation... \star/
00105
           for (ip = 0; ip < atm->np; ip++) {
            zm += Z(atm->p[ip]) / atm->np;
lonm += atm->lon[ip] / atm->np;
latm += atm->lat[ip] / atm->np;
00106
00107
00108
             zs += gsl_pow_2(Z(atm->p[ip])) / atm->np;
             lons += gsl_pow_2(atm->lon[ip]) / atm->np;
lats += gsl_pow_2(atm->lat[ip]) / atm->np;
00110
00111
00112
00113
          /* Normalize... */
zs = sqrt(zs - gsl_pow_2(zm));
00114
00115
          lons = sqrt(lons - gsl_pow_2(lonm));
lats = sqrt(lats - gsl_pow_2(latm));
00116
00117
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
00123
00124
           /\star Get time from filename... \star/
           sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00125
           vear = atoi(tstr);
00126
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00127
           mon = atoi(tstr);
00128
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]);
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]),
                                       Z(atm->p[atm->np - atm->np / 10]),
Z(atm->p[atm->np - atm->np / 4]),
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],
atm->lon[atm->np / 4], atm->lon[atm->np / 2],
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],
atm->lon[atm->np - 1],
latm, lats, atm->lat[0], atm->lat[atm->np / 10],
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]);
00147
00148
00149
00150
00151
00152
00153
00154
00155
                /* Close file... */
00156
00157
                fclose(out);
00158
                /* Free... */
00160
                free(atm);
00161
00162
                return EXIT_SUCCESS;
00163 }
```

## 5.3 cluster.c File Reference

Clustering of trajectories.

#### **Functions**

• int main (int argc, char \*argv[])

#### 5.3.1 Detailed Description

Clustering of trajectories.

Definition in file cluster.c.

#### 5.3.2 Function Documentation

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

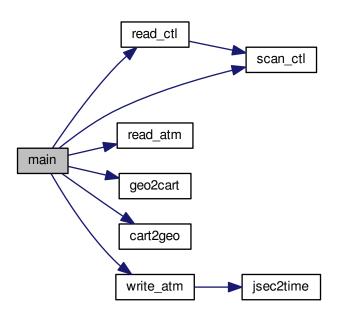
Definition at line 41 of file cluster.c.

```
00044
00045
        ctl_t ctl;
00046
00047
        atm_t *atm;
00048
00049
        gsl rng *rng;
00050
00051
        FILE *out;
00052
        static double d2, *dist, lat, lon, rmsd[NS],
x[3], xs[NT][NS][3], z, zs[NT][NS];
00053
00054
00055
00056
        static int *cluster, f, idx[NS], ip, is, it, itmax, np[NS], ns;
00057
00058
        /* Check arguments... */
        if (argc < 4)
00059
00060
          ERRMSG("Give parameters: <ctl> <cluster.log> <atml> [<atm2> <atm3> ...]");
00061
00062
        /* Read control parameters... */
```

```
read_ctl(argv[1], argc, argv, &ctl);
00064
        ns = (int) scan_ctl(argv[1], argc, argv, "CLUSTER_NS", -1, "7", NULL);
00065
        if (ns > NS)
         ERRMSG("Too many seeds!");
00066
00067
        itmax =
00068
          (int) scan_ctl(argv[1], argc, argv, "CLUSTER_ITMAX", -1, "10", NULL);
00070
        /* Initialize random number generator... */
00071
        gsl_rng_env_setup();
00072
        rng = gsl_rng_alloc(gsl_rng_default);
00073
        /* Allocate... */
00074
        ALLOC(atm, atm_t, 1);
00075
00076
        ALLOC(cluster, int,
00077
             NP);
00078
        ALLOC(dist, double,
00079
             NP * NS);
08000
00081
       /* Create output file... */
        printf("Write cluster data: %s\n", argv[2]);
00082
00083
        if (!(out = fopen(argv[2], "w")))
00084
         ERRMSG("Cannot create file!");
00085
00086
        /* Write header... */
00087
        fprintf(out,
                "# $1 = iteration index\n"
00088
00089
                "# $2 = seed index \n"
00090
                "# $3 = time step index \n"
                "# $4 = mean altitude [km]\n"
00091
                "# $5 = mean longitude [deg] \n"
00092
                "# $6 = mean latitude [deg]\n"
"# $7 = number of points\n" "# $8 = RMSD [km^2]\n");
00093
00094
00095
00096
       /\star Get seeds (random selection of trajectories)... \star/
00097
       for (f = 3; f < argc; f++) {</pre>
00098
00099
         /* Check number of timesteps... */
if (f - 3 > NT)
00101
            ERRMSG("Too many timesteps!");
00102
00103
          /* Read atmopheric data... */
00104
          read_atm(argv[f], &ctl, atm);
00105
00106
          /\star Pick seeds (random selection)... \star/
          if (f == 3)
00107
00108
           for (is = 0; is < ns; is++)</pre>
00109
              idx[is] = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00110
00111
          /* Save seeds... */
          for (is = 0; is < ns; is++) {</pre>
00112
           geo2cart(0, atm->lon[idx[is]], atm->lat[idx[is]], xs[f - 3][is]);
00113
00114
            zs[f - 3][is] = Z(atm->p[idx[is]]);
00115
00116
       }
00117
        /* Iterations... */
for (it = 0; it < itmax; it++) {
00118
00120
00121
          /* Write output... */
          00122
00123
00124
00125
00126
00127
00128
00129
          }
00130
00131
          /* Init... */
          for (ip = 0; ip < atm->np; ip++)
00132
00133
           for (is = 0; is < ns; is++) {
00134
             dist[ip * NS + is] = 0;
             rmsd[is] = 0;
00135
00136
00137
00138
          /* Get distances between seeds and trajectories... */
00139
          for (f = 3; f < argc; f++) {</pre>
00140
00141
            /* Read atmopheric data... */
00142
            read_atm(argv[f], &ctl, atm);
00143
00144
            /* Get distances... */
00145
            for (ip = 0; ip < atm->np; ip++) {
00146
              geo2cart(0, atm->lon[ip], atm->lat[ip], x);
00147
              z = Z(atm->p[ip]);
00148
              for (is = 0; is < ns; is++) {
00149
                d2 =
```

```
DIST2(x, xs[f - 3][is]) + gsl_pow_2((z - zs[f - 3][is]) * 200.);
00151
                  dist[ip * NS + is] += d2;
00152
                  rmsd[is] += d2;
00153
00154
             }
           }
00155
00156
00157
           /* Assign clusters... */
           for (ip = 0; ip < atm->np; ip++)
  cluster[ip] = (int) gsl_stats_min_index(&dist[ip * NS], 1, (size_t) ns);
00158
00159
00160
00161
           /* Recalculate seeds (mean trajectories)... */
00162
           for (f = 3; f < argc; f++) {</pre>
00163
00164
             /∗ Read atmopheric data... ∗/
00165
             read_atm(argv[f], &ctl, atm);
00166
00167
             /* Calculate new seeds... */
             for (is = 0; is < ns; is++) {</pre>
00168
              xs[f - 3][is][0] = 0;
00169
00170
               xs[f - 3][is][1] = 0;
               xs[f - 3][is][2] = 0;
00171
               zs[f - 3][is] = 0;
00172
00173
               np[is] = 0;
00174
00175
             for (ip = 0; ip < atm->np; ip++) {
00176
               geo2cart(0, atm->lon[ip], atm->lat[ip], x);
               xs[f - 3][cluster[ip]][0] += x[0];
xs[f - 3][cluster[ip]][1] += x[1];
00177
00178
               xs[f - 3][cluster[ip]][2] += x[2];
zs[f - 3][cluster[ip]] += Z(atm->p[ip]);
00179
00180
00181
               np[cluster[ip]]++;
00182
00183
             for (is = 0; is < ns; is++) {</pre>
              xs[f - 3][is][0] /= np[is];
xs[f - 3][is][1] /= np[is];
00184
00185
               xs[f - 3][is][2] /= np[is];
00186
                zs[f - 3][is] /= np[is];
00187
00188
00189
00190
00191
        /* Write output... */
for (is = 0; is < ns; is++) {
  fprintf(out, "\n");
  for (f = 3; f < argc; f++) {</pre>
00192
00193
00194
00195
             00196
00197
00198
00199
00200
        }
00201
00202
        /* Close output file... */
00203
        fclose(out);
00204
00205
         /\star Write clustering results... \star/
00206
        if (ctl.qnt_ens >= 0)
00207
00208
           /\star Recalculate seeds (mean trajectories)... \star/
00209
           for (f = 3; f < argc; f++) {</pre>
00210
00211
             /* Read atmopheric data... */
00212
             read_atm(argv[f], &ctl, atm);
00213
00214
             /* Set ensemble ID... */
             for (ip = 0; ip < atm->np; ip++)
  atm->q[ctl.qnt_ens][ip] = cluster[ip];
00215
00216
00217
00218
             /* Write atmospheric data... *
             write_atm(argv[f], &ctl, atm, 0);
00220
00221
00222
        /* Free... */
        gsl_rng_free(rng);
00223
00224
         free (atm);
00225
        free(cluster);
00226
        free(dist);
00227
00228
         return EXIT_SUCCESS;
00229 }
```

Here is the call graph for this function:



#### 5.4 cluster.c

```
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.
00008
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
         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) 2013-2018 Forschungszentrum Juelich GmbH
00017
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* --
00028
          Defines...
00029
00030
00032 #define NS 100
00033
00035 #define NT 1000
00036
00037 /* -----
00038
          Main...
00039
00040
00041 int main(
00042 int argc,
00043 char *arg
         char *argv[]) {
00044
00045
         ctl_t ctl;
00046
```

5.4 cluster.c 33

```
00047
        atm_t *atm;
00048
00049
        gsl_rng *rng;
00050
00051
        FILE *out:
00052
        static double d2, *dist, lat, lon, rmsd[NS],
00054
          x[3], xs[NT][NS][3], z, zs[NT][NS];
00055
00056
        static int *cluster, f, idx[NS], ip, is, it, itmax, np[NS], ns;
00057
00058
        /* Check arguments... */
00059
        if (argc < 4)
00060
          ERRMSG("Give parameters: <ctl> <cluster.log> <atm1> [<atm2> <atm3> ...]");
00061
        /* Read control parameters... */
00062
        read_ctl(argv[1], argc, argv, &ctl);
ns = (int) scan_ctl(argv[1], argc, argv, "CLUSTER_NS", -1, "7", NULL);
00063
00064
        if (ns > NS)
00065
00066
          ERRMSG("Too many seeds!");
00067
           (int) scan_ctl(argv[1], argc, argv, "CLUSTER_ITMAX", -1, "10", NULL);
00068
00069
00070
        /* Initialize random number generator... */
00071
        gsl_rng_env_setup();
00072
        rng = gsl_rng_alloc(gsl_rng_default);
00073
00074
         /* Allocate... */
00075
        ALLOC(atm, atm_t, 1);
00076
        ALLOC(cluster, int,
00077
              NP);
00078
        ALLOC(dist, double,
00079
              NP * NS);
08000
        /* Create output file... */
printf("Write cluster data: %s\n", argv[2]);
if (!(out = fopen(argv[2], "w")))
00081
00082
00083
          ERRMSG("Cannot create file!");
00084
00085
00086
         /* Write header... */
        00087
00088
                 "# $2 = seed index\n"
00089
00090
                 "# $3 = time step index\n'
00091
                 "# $4 = mean altitude [km] \n"
00092
                 "# $5 = mean longitude [deg]\n"
                 "# $6 = mean latitude [deg]\n"  
"# $7 = number of points\n"  
"# $8 = RMSD [km^2]\n");
00093
00094
00095
00096
        /* Get seeds (random selection of trajectories)... */
        for (f = 3; f < argc; f++) {</pre>
00097
00098
00099
           /\star Check number of timesteps... \star/
00100
          if (f - 3 > NT)
             ERRMSG("Too many timesteps!");
00101
00102
           /* Read atmopheric data... */
00104
          read_atm(argv[f], &ctl, atm);
00105
00106
           /* Pick seeds (random selection)... */
          if (f == 3)
  for (is = 0; is < ns; is++)</pre>
00107
00108
00109
               idx[is] = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00110
00111
           /* Save seeds... */
00112
          for (is = 0; is < ns; is++) {</pre>
            geo2cart(0, atm->lon[idx[is]], atm->lat[idx[is]], xs[f - 3][is]);
zs[f - 3][is] = Z(atm->p[idx[is]]);
00113
00114
00115
00116
        }
00117
        /* Iterations... */
for (it = 0; it < itmax; it++) {</pre>
00118
00119
00120
00121
           /* Write output... */
00122
          for (is = 0; is < ns; is++) {
00123
             fprintf(out, "\n");
             for (f = 3; f < argc; f++) {
00124
               00125
00126
00127
00128
            }
00129
00130
          /* Init... */
for (ip = 0; ip < atm->np; ip++)
for (is = 0; is < ns; is++) {
00131
00132
00133
```

```
00134
                dist[ip * NS + is] = 0;
00135
                rmsd[is] = 0;
00136
00137
00138
            /* Get distances between seeds and trajectories... */
00139
            for (f = 3; f < argc; f++) {
00141
              /* Read atmopheric data... */
00142
              read_atm(argv[f], &ctl, atm);
00143
00144
              /* Get distances... */
00145
              for (ip = 0; ip < atm->np; ip++) {
                geo2cart(0, atm->lon[ip], atm->lat[ip], x);
00146
00147
                 z = Z(atm->p[ip]);
00148
                 for (is = 0; is < ns; is++) {</pre>
00149
                  d2 =
                   DIST2(x, xs[f - 3][is]) + gsl_pow_2((z - zs[f - 3][is]) * 200.);
dist[ip * NS + is] += d2;
rmsd[is] += d2;
00150
00151
00152
00153
                }
00154
00155
            }
00156
            /* Assign clusters... */
for (ip = 0; ip < atm->np; ip++)
  cluster[ip] = (int) gsl_stats_min_index(&dist[ip * NS], 1, (size_t) ns);
00157
00158
00159
00160
            /* Recalculate seeds (mean trajectories)... */ for (f = 3; f < argc; f++) {
00161
00162
00163
00164
              /* Read atmopheric data... */
00165
              read_atm(argv[f], &ctl, atm);
00166
00167
              /* Calculate new seeds... */
              for (is = 0; is < ns; is++) {
  xs[f - 3][is][0] = 0;</pre>
00168
00169
                 xs[f - 3][is][1] = 0;
00170
                xs[f - 3][is][2] = 0;
00172
                 zs[f - 3][is] = 0;
00173
                np[is] = 0;
00174
00175
              for (ip = 0; ip < atm->np; ip++) {
                geo2cart(0, atm->lon[ip], atm->lat[ip], x);
xs[f - 3][cluster[ip]][0] += x[0];
00176
00177
00178
                 xs[f - 3][cluster[ip]][1] += x[1];
                 xs[f - 3][cluster[ip]][2] += x[2];
zs[f - 3][cluster[ip]] += Z(atm->p[ip]);
00179
00180
00181
                 np[cluster[ip]]++;
00182
00183
              for (is = 0; is < ns; is++) {
                xs[f - 3][is][0] /= np[is];
xs[f - 3][is][1] /= np[is];
00184
00185
                 xs[f - 3][is][2] /= np[is];
00186
00187
                zs[f - 3][is] /= np[is];
00188
00189
           }
00190
00191
00192
          /* Write output... */
         for (is = 0; is < ns; is++) {
  fprintf(out, "\n");
  for (f = 3; f < argc; f++) {
    cart2geo(xs[f - 3][is], &z, &lon, &lat);
}</pre>
00193
00194
00195
00196
              fprintf(out, "%d %d %d %g %g %g %d %g\n", it, is, f - 3, zs[f - 3][is], lon, lat, np[is], rmsd[is]);
00197
00198
00199
00200
00201
00202
         /* Close output file... */
00203
         fclose(out);
00204
00205
         /* Write clustering results... */
00206
         if (ctl.qnt_ens >= 0)
00207
00208
            /* Recalculate seeds (mean trajectories)... */
00209
           for (f = 3; f < argc; f++) {
00210
00211
              /* Read atmopheric data... */
00212
              read_atm(argv[f], &ctl, atm);
00213
00214
              /* Set ensemble ID... */
              for (ip = 0; ip < atm->np; ip++)
00215
00216
                atm->q[ctl.qnt_ens][ip] = cluster[ip];
00217
00218
              /\star Write atmospheric data... \star/
00219
              write_atm(argv[f], &ctl, atm, 0);
00220
```

5.5 conv.c File Reference 35

```
00221
00222  /* Free... */
00223  gsl_rng_free(rng);
00224  free(atm);
00225  free(cluster);
00226  free(dist);
00227
00228  return EXIT_SUCCESS;
00229 }
```

#### 5.5 conv.c File Reference

Convert file format of atmospheric data files.

#### **Functions**

• int main (int argc, char \*argv[])

## 5.5.1 Detailed Description

Convert file format of atmospheric data files.

Definition in file conv.c.

## 5.5.2 Function Documentation

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

Definition at line 27 of file conv.c.

```
00029
00030
00031
      ctl_t ctl;
00032
00033
      atm_t *atm;
00034
00035
       /* Check arguments... */
      if (argc < 6)
00036
       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... */
00051
      ctl.atm_type = atoi(argv[5]);
00052
      write_atm(argv[4], &ctl, atm, 0);
00053
      /* Free... */
00054
00055
      free(atm);
00056
       return EXIT_SUCCESS;
00058 }
```

Here is the call graph for this function:

## 5.6 conv.c

```
00001 /*
00002
        This file is part of MPTRAC.
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 along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00015
00016
00017
        Copyright (C) 2013-2018 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)</pre>
         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... */
00051
        ctl.atm_type = atoi(argv[5]);
00052
        write_atm(argv[4], &ctl, atm, 0);
00053
00054
        /* Free... */
00055
        free(atm);
00056
00057
        return EXIT SUCCESS:
00058 }
```

# 5.7 day2doy.c File Reference

Convert date to day of year.

**Functions** 

• int main (int argc, char \*argv[])

# 5.7.1 Detailed Description

Convert date to day of year.

Definition in file day2doy.c.

5.8 day2doy.c 37

#### 5.7.2 Function Documentation

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

Definition at line 27 of file day2doy.c.

```
00029
                        {
00030
00031
        int day, doy, mon, year;
00032
00033
        /* Check arguments... */
00034
        if (argc < 4)
00035
          ERRMSG("Give parameters: <year> <mon> <day>");
00036
00037
        /* Read arguments... */
00038
        year = atoi(argv[1]);
        mon = atoi(argv[2]);
00039
00040
        day = atoi(argv[3]);
00041
        /* Convert... */
00042
       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:

# 5.8 day2doy.c

```
00001 /*
00002
        This file is part of MPTRAC.
00003
00004
        MPTRAC is free software: you can redistribute it and/or modify
00005
        it under the terms of the GNU General Public License as published by
00006
        the Free Software Foundation, either version 3 of the License, or
00007
        (at your option) any later version.
00008
00009
        MPTRAC is distributed in the hope that it will be useful,
00010
        but WITHOUT ANY WARRANTY; without even the implied warranty of
00011
        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
        GNU General Public License for more details.
00013
00014
        You should have received a copy of the GNU General Public License 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
        int day, doy, mon, year;
00032
00033
        /* Check arguments... ∗/
00034
        if (argc < 4)
00035
          ERRMSG("Give parameters: <year> <mon> <day>");
00036
00037
        /* Read arguments... */
00038
        year = atoi(argv[1]);
        mon = atoi(argv[2]);
00039
00040
        day = atoi(argv[3]);
00041
00042
         /* Convert... */
00043
        day2doy(year, mon, day, &doy);
00044
        printf("%d %d\n", year, doy);
00045
00046
        return EXIT_SUCCESS;
00047 }
```

# 5.9 dist.c File Reference

Calculate transport deviations of trajectories.

## **Functions**

• int main (int argc, char \*argv[])

## 5.9.1 Detailed Description

Calculate transport deviations of trajectories.

Definition in file dist.c.

#### 5.9.2 Function Documentation

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

Definition at line 27 of file dist.c.

```
00029
                        {
00030
00031
        ctl_t ctl;
00032
00033
        atm_t *atm1, *atm2;
00034
00035
        FILE *out;
00036
00037
        char tstr[LEN];
00038
00039
        double ahtd, aqtd[NQ], atcel[NQ], atce2[NQ], avtd, lat0, lat1,
00040
          *lat1_old, *lat2_old, *lh1, *lh2, lon0, lon1, *lon1_old, *lon2_old,
          *lv1, *lv2, p0, p1, *q1, *q2, rhtd, rqtd[NQ], rtcel[NQ], rtce2[NQ], rvtd, t, t0 = 0, x0[3], x1[3], x2[3], z1, *z1_old, z2, *z2_old;
00041
00042
00043
00044
        int ens, f, ip, iq, np, year, mon, day, hour, min;
00045
00046
         /* Allocate... */
00047
        ALLOC(atm1, atm_t, 1);
00048
        ALLOC(atm2, atm_t, 1);
00049
        ALLOC(lon1_old, double,
00050
              NP);
00051
        ALLOC(lat1_old, double,
00052
               NP);
        ALLOC(z1_old, double,
00053
00054
              NP);
        ALLOC(lh1, double,
00055
              NP);
00056
00057
        ALLOC(lv1, double,
00058
               NP);
        ALLOC(lon2_old, double,
00059
        NP);
ALLOC(lat2_old, double,
00060
00061
00062
               NP);
00063
        ALLOC(z2_old, double,
00064
               NP);
        ALLOC(1h2, double,
00065
00066
              NP);
        ALLOC(1v2, double,
00067
00068
              NP);
        ALLOC(q1, double,
00069
00070
              NQ * NP);
        ALLOC(q2, double, NQ * NP);
00071
00072
00073
00074
        /* Check arguments... */
00075
        if (argc < 5)
          ERRMSG("Give parameters: <ctl> <outfile> <atmla> <atmlb>"
```

```
" [<atm2a> <atm2b> ...]");
00078
00079
           /* Read control parameters... */
08000
          read_ctl(argv[1], argc, argv, &ctl);
          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_Z0", -1, "-1000", NULL));
p1 = P(scan_ctl(argv[1], argc, argv, "DIST_Z1", -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);
00081
00082
00084
00085
00086
00087
00088
00089
          /* Write info... */
00090
          printf("Write transport deviations: %s\n", argv[2]);
00091
          /* Create output file... */
if (!(out = fopen(argv[2], "w")))
00092
00093
            ERRMSG("Cannot create file!");
00094
00095
00096
          /* Write header... */
00097
          fprintf(out,
00098
                      "# $1 = time [s] \n"
                      "# $2 = trajectory time [s] \n"
00099
                     "# $4 = AHTD [km]\n"
"# $4 = RHTD [%%]\n" "# $5 = AVTD [km]\n" "# $6 = RVTD [%%]\n");
00100
00101
          for (iq = 0; iq < ctl.nq; iq++)</pre>
00102
00103
            fprintf(out,
                        "# $%d = AQTD (%s) [%s]\n"
"# $%d = RQTD (%s) [%s]\n",
00104
00105
          7 + 2 * iq, ctl.qnt_name[iq], ctl.qnt_unit[iq], 8 + 2 * iq, ctl.qnt_name[iq]); for (iq = 0; iq < ctl.nq; iq++)
00106
00107
00108
00109
           fprintf(out,
00110
                         "# \$%d = ATCE_1 (%s) [%s]\n"
                        "# $%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]);
00111
00112
00113
00114
          for (iq = 0; iq < ctl.nq; iq++)</pre>
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
00120
          fprintf(out, "# \$%d = number of particles\n\n", 7 + 6 * ctl.nq);
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)
               ERRMSG("Different numbers of parcels!");
00131
             for (ip = 0; ip < atm1->np; ip++)
00132
               if (gsl_finite(atm1->time[ip]) && gsl_finite(atm2->time[ip])
00134
                      && atm1->time[ip] != atm2->time[ip])
00135
                   ERRMSG("Times do not match!");
00136
             /* Get time from filename... */
00137
             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]);
00143
             day = atoi(tstr);
sprintf(tstr, "%.2s", &arqv[f][strlen(arqv[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
             /* Save initial data... */
00150
             if (f == 3) {
00151
               t0 = t;
00152
00153
                for (iq = 0; iq < ctl.nq; iq++)</pre>
                   for (ip = 0; ip < atm1->np; ip++) {
  q1[iq * NP + ip] = atm1->q[iq][ip];
  q2[iq * NP + ip] = atm2->q[iq][ip];
00154
00155
00156
00157
00158
             }
00159
00160
             /* Init... */
00161
             np = 0;
             ahtd = avtd = rhtd = rvtd = 0;
00162
             for (iq = 0; iq < ctl.nq; iq++)</pre>
00163
```

```
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... */
              if (!gsl_finite(atm1->time[ip]) || !gsl_finite(atm2->time[ip]))
00170
00171
00172
00173
               /\star Check ensemble ID... \star/
              if (ens >= 0 && ctl.qnt_ens >= 0 && atml->q[ctl.qnt_ens][ip] != ens)
00174
00175
                 continue;
               if (ens >= 0 && ctl.qnt_ens >= 0 && atm2->q[ctl.qnt_ens][ip] != ens)
00176
00177
00178
00179
               /* Check spatial range... */
              if (atm1->p[ip] > p0 || atm1->p[ip] < p1
    || atm1->lon[ip] < lon0 || atm1->lon[ip] > lon1
    || atm1->lat[ip] < lat0 || atm1->lat[ip] > lat1)
00180
00181
00182
00183
00184
               if (atm2->p[ip] > p0 || atm2->p[ip] < p1
                   00185
00186
00187
                 continue:
00188
00189
               /* Convert coordinates... */
00190
              geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00191
               geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00192
               z1 = Z(atm1->p[ip]);
              z2 = Z(atm2->p[ip]);
00193
00194
00195
               /* Calculate absolute transport deviations... */
00196
              ahtd += DIST(x1, x2);
00197
               avtd += fabs(z1 - z2);
               for (iq = 0; iq < ctl.nq; iq++)
00198
                 aqtd[iq] += fabs(atm1->q[iq][ip] - atm2->q[iq][ip]);
00199
00200
               /* Calculate relative transport deviations... */
00202
00203
00204
                 /\star Get trajectory lengths... \star/
                 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
00209
                 geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
                 lh2[ip] += DIST(x0, x2);
lv2[ip] += fabs(z2_old[ip] - z2);
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]);
for (iq = 0; iq < ctl.nq; iq++)
  rqtd[iq] += 200. * fabs(atml->q[iq][ip] - atm2->q[iq][ip])
00217
00218
00219
                      / (fabs(atm1->q[iq][ip]) + fabs(atm2->q[iq][ip]));
00221
00222
                 /* Get tracer conservation errors... */
00223
                 for (iq = 0; iq < ctl.nq; iq++)</pre>
                   or (iq = 0; iq < cti.iq; iq+r) ;

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]));

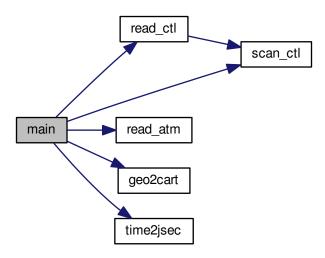
atce2[iq] += fabs(atm2->q[iq][ip] - q2[iq * NP + ip]);

rtce2[iq] += 200. * fabs(atm2->q[iq][ip] - q2[iq * NP + ip])
00224
00225
00226
00227
00228
                      / (fabs(atm2->q[iq][ip]) + fabs(q2[iq * NP + ip]));
00229
00230
00231
              }
00232
00233
               /* Save positions of air parcels... */
              lon1_old[ip] = atm1->lon[ip];
lat1_old[ip] = atm1->lat[ip];
00234
00235
00236
              z1\_old[ip] = z1;
00237
00238
               lon2 old[ip] = atm2->lon[ip];
               lat2_old[ip] = atm2->lat[ip];
00239
00240
              z2\_old[ip] = z2;
00241
00242
               /\star Increment air parcel counter... \star/
00243
              np++;
00244
00245
            00246
00247
                     ahtd / np, rhtd / np, avtd / np, rvtd / np);
00248
            for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");</pre>
00249
00250
```

5.10 dist.c 41

```
fprintf(out, ctl.qnt_format[iq], aqtd[iq] / np);
fprintf(out, " ");
fprintf(out, ctl.qnt_format[iq], rqtd[iq] / np);
00252
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
00258
00259
               fprintf(out, ctl.qnt_format[iq], rtcel[iq] / np);
00260
             for (iq = 0; iq < ctl.nq; iq++) {
    fprintf(out, " ");
    fprintf(out, ctl.qnt_format[iq], atce2[iq] / np);
    fprintf(out, " ");</pre>
00261
00262
00263
00264
00265
                fprintf(out, ctl.qnt_format[iq], rtce2[iq] / np);
00266
             fprintf(out, " %d\n", np);
00267
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);
          free(lon2_old);
free(lat2_old);
00281
00282
00283
          free(z2_old);
00284
          free(lh2);
00285
          free(lv2);
00286
00287
          return EXIT_SUCCESS;
00288 }
```

Here is the call graph for this function:



#### 5.10 dist.c

```
00001 /\star 00002 This file is part of MPTRAC. 00003
```

```
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,
          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
         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) 2013-2018 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028
         int argc,
00029
         char *argv[]) {
00030
00031
         ctl t ctl;
00032
00033
         atm_t *atm1, *atm2;
00034
00035
         FILE *out;
00036
00037
          char tstr[LEN];
00038
00039
          double ahtd, aqtd[NQ], atcel[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,
00040
00041
00042
            t, t0 = 0, x0[3], x1[3], x2[3], z1, *z1_old, z2, *z2_old;
00043
00044
          int ens, f, ip, iq, np, year, mon, day, hour, min;
00045
00046
          /* Allocate... */
00047
          ALLOC(atm1, atm_t, 1);
00048
          ALLOC(atm2, atm_t,
00049
          ALLOC(lon1_old, double,
00050
                 NP);
00051
          ALLOC(lat1 old, double,
00052
                 NP);
          ALLOC(z1_old, double,
00053
00054
                 NP);
00055
         ALLOC(lh1, double,
00056
                 NP);
         ALLOC(lv1, double,
00057
00058
                 NP);
         ALLOC(lon2_old, double,
00059
00060
                  NP);
00061
          ALLOC(lat2_old, double,
00062
                 NP);
00063
          ALLOC(z2_old, double,
00064
                  NP);
          ALLOC(lh2, double,
00065
00066
                  NP);
00067
          ALLOC(1v2, double,
00068
                 NP);
         ALLOC(q1, double, NQ * NP);
00069
00070
00071
          ALLOC(q2, double,
00072
                 NQ * NP);
00073
00074
          /* Check arguments... */
00075
          if (argc < 5)
00076
            ERRMSG("Give parameters: <ctl> <outfile> <atmla> <atmlb>"
                      " [<atm2a> <atm2b> ...]");
00077
00078
00079
          /* 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_20", -1, "-1000", NULL));
p1 = P(scan_ct1(argv[1], argc, argv, "DIST_21", -1, "1000", NULL));
lat0 = scan_ct1(argv[1], argc, argv, "DIST_LAT0", -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_LON1", -1, "1000", NULL);
00081
00082
00083
00084
00085
00086
00087
00088
00089
          /* Write info... */
          printf("Write transport deviations: %s\n", argv[2]);
00090
00091
00092
          /* Create output file...
          if (!(out = fopen(argv[2], "w")))
00093
            ERRMSG("Cannot create file!");
00094
00095
```

5.10 dist.c 43

```
/* Write header... */
00097
         fprintf(out,
00098
                   "# $1 = time [s] \n"
                   "# $2 = trajectory time [s]\n"
00099
                   "# $3 = AHTD [km]\n"
"# $4 = RHTD [%%]\n" "# $5 = AVTD [km]\n" "# $6 = RVTD [%%]\n");
00100
00101
         for (iq = 0; iq < ctl.nq; iq++)</pre>
00102
00103
           fprintf(out,
00104
                     "# $%d = AQTD (%s) [%s]\n"
                     "# $%d = RQTD (%s) [%%]\n",
00105
                     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,
00110
                     "# $%d = ATCE_1 (%s) [%s]\n"
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,
                     "# $%d = ATCE_2 (%s) [%s]\n"
"# $%d = RTCE_2 (%s) [%%]\n",
00116
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]); \\ fprintf(out, "# $%d = number of particles\n\n", 7 + 6 * ctl.nq);
00118
00119
00120
00121
         /* Loop over file pairs... */
00122
00123
         for (f = 3; f < argc; f += 2) {</pre>
00124
00125
            /* Read atmopheric data... */
00126
           read_atm(argv[f], &ctl, atm1);
read_atm(argv[f + 1], &ctl, atm2);
00127
00128
            /* Check if structs match... */
00129
00130
            if (atm1->np != atm2->np)
             ERRMSG("Different numbers of parcels!");
00131
00132
            for (ip = 0; ip < atm1->np; ip++)
             if (gsl_finite(atm1->time[ip]) && gsl_finite(atm2->time[ip])
00133
00134
                   && atm1->time[ip] != atm2->time[ip])
00135
                ERRMSG("Times do not match!");
00136
00137
            /* 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);
sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00143
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
00153
              for (iq = 0; iq < ctl.nq; iq++)</pre>
                for (ip = 0; ip < atml->np; ip++) {
    q1[iq * NP + ip] = atml->q[iq][ip];
    q2[iq * NP + ip] = atm2->q[iq][ip];
00154
00155
00156
00157
00158
           }
00159
00160
            /* Init... */
00161
           np = 0;
00162
            ahtd = avtd = rhtd = rvtd = 0;
            for (iq = 0; iq < ctl.nq; iq++)</pre>
00163
             aqtd[iq] = atce1[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... */
00170
              if (!gsl finite(atm1->time[ip]) || !gsl finite(atm2->time[ip]))
00171
                continue;
00172
              /* Check ensemble ID... */
00173
00174
              if (ens >= 0 && ctl.qnt_ens >= 0 && atm1->q[ctl.qnt_ens][ip] != ens)
00175
                continue:
              if (ens >= 0 && ctl.qnt_ens >= 0 && atm2->q[ctl.qnt_ens][ip] != ens)
00176
00177
                continue;
00178
00179
              /* Check spatial range... */
00180
              if (atm1->p[ip] > p0 || atm1->p[ip] < p1
                   || atml->lon[ip] < lon0 || atml->lon[ip] > lon1
|| atml->lat[ip] < lat0 || atml->lat[ip] > lat1)
00181
00182
```

```
00183
                   continue;
                if (atm2->p[ip] > p0 || atm2->p[ip] < p1
00184
00185
                      || atm2 -> lon[ip] < lon0 || atm2 -> lon[ip] > lon1
                     || atm2->lat[ip] < lat0 || atm2->lat[ip] > lat1)
00186
00187
                   continue;
00188
00189
                /* Convert coordinates... */
00190
                geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00191
                geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00192
                z1 = Z(atm1->p[ip]);
                z2 = Z(atm2->p[ip]);
00193
00194
00195
                /* Calculate absolute transport deviations... */
00196
                ahtd += DIST(x1, x2);
                avtd += fabs(z1 - z2);
for (iq = 0; iq < ctl.nq; iq++)
00197
00198
                   aqtd[iq] += fabs(atm1->q[iq][ip] - atm2->q[iq][ip]);
00199
00200
00201
                /* Calculate relative transport deviations... */
00202
                if (f > 3) {
00203
00204
                   /* Get trajectory lengths... */
                   geo2cart(0, lon1_old[ip], lat1_old[ip], x0);
lh1[ip] += DIST(x0, x1);
00205
00206
00207
                   lv1[ip] += fabs(z1_old[ip] - z1);
00208
00209
                   geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
                   lh2[ip] += DIST(x0, x2);
lv2[ip] += fabs(z2_old[ip] - z2);
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]);
for (iq = 0; iq < ctl.nq; iq++)
  rqtd[iq] += 200. * fabs(atml->q[iq][ip] - atm2->q[iq][ip])
00217
00218
00219
                         / (fabs(atm1->q[iq][ip]) + fabs(atm2->q[iq][ip]));
00221
00222
                   /* Get tracer conservation errors... */
                  /* Get tracer conservation errors... */
for (iq = 0; iq < ctl.nq; iq++) {
    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]));
    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]));
00223
00224
00225
00226
00227
00228
00229
00230
                  }
00231
00232
                /* Save positions of air parcels... */
                lon1_old[ip] = atml->lon[ip];
lat1_old[ip] = atml->lat[ip];
00234
00235
00236
                z1\_old[ip] = z1;
00237
00238
                lon2_old[ip] = atm2->lon[ip];
lat2_old[ip] = atm2->lat[ip];
00239
00240
                z2_old[ip] = z2;
00241
00242
                /\star Increment air parcel counter... \star/
00243
               np++;
00244
00245
00246
             /* Write output... */
             00247
00248
             for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl.qnt_format[iq], aqtd[iq] / np);
  fprintf(out, " ");</pre>
00249
00250
00251
00252
00253
                fprintf(out, ctl.qnt_format[iq], rqtd[iq] / np);
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
00259
                fprintf(out, ctl.qnt_format[iq], rtcel[iq] / np);
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, " dn", np);
00268
00269
```

```
/* Close file... */
00271
       fclose(out);
00272
00273
       /* Free... */
00274
       free(atm1);
00275
       free(atm2);
       free(lon1_old);
00277
       free(lat1_old);
00278
       free(z1_old);
00279
       free(lh1);
00280
       free(lv1);
       free(lon2_old);
00281
00282
       free(lat2_old);
00283
       free(z2_old);
00284
       free(lh2);
00285
       free(lv2);
00286
00287
       return EXIT_SUCCESS;
00288 }
```

# 5.11 doy2day.c File Reference

Convert day of year to date.

#### **Functions**

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

## 5.11.1 Detailed Description

Convert day of year to date.

Definition in file doy2day.c.

# 5.11.2 Function Documentation

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

Definition at line 27 of file doy2day.c.

```
00030
00031
         int day, doy, mon, year;
00032
00033
         /* Check arguments... */
if (argc < 3)</pre>
00034
           ERRMSG("Give parameters: <year> <doy>");
00036
00037
         /* Read arguments... */
        year = atoi(argv[1]);
doy = atoi(argv[2]);
00038
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:

# 5.12 doy2day.c

```
00001 /*
        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 the Free Software Foundation, either version 3 of the License, or
00005
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
        {\tt 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
        int day, doy, mon, year;
00032
00033
        /* Check arguments... */
00034
        if (argc < 3)
00035
          ERRMSG("Give parameters: <year> <doy>");
00036
00037
        /* Read arguments... */
00038
        year = atoi(argv[1]);
00039
        doy = atoi(argv[2]);
00040
00041
        /* Convert... */
        doy2day(year, doy, &mon, &day);
printf("%d %d %d\n", year, mon, day);
00042
00043
00044
00045
        return EXIT_SUCCESS;
00046 }
```

## 5.13 extract.c File Reference

Extract single trajectory from atmospheric data files.

## **Functions**

• int main (int argc, char \*argv[])

## 5.13.1 Detailed Description

Extract single trajectory from atmospheric data files.

Definition in file extract.c.

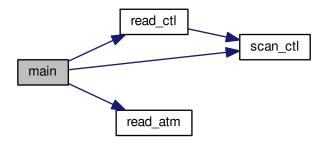
#### 5.13.2 Function Documentation

## 5.13.2.1 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> <atml> [<atm2> ...]");
00045
00046
        /* Read control parameters... */
        read_ctl(argv[1], argc, argv, &ctl);
ip = (int) scan_ctl(argv[1], argc, argv, "EXTRACT_IP", -1, "0", NULL);
00047
00048
00049
00050
        /* Write info... */
00051
        printf("Write trajectory data: %s\n", 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
                 "# $1 = time [s]\n"
                 "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00060
00061
        for (iq = 0; iq < ctl.nq; iq++)
fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00062
00063
        ctl.qnt_unit[iq]);
fprintf(out, "\n");
00064
00065
00066
00067
        /* Loop over files... */
00068
        for (f = 3; f < argc; f++) {</pre>
00069
          /* Read atmopheric data... */
if (!(in = fopen(argv[f], "r")))
00070
00071
00072
            continue;
00073
          else
00074
            fclose(in);
00075
          read_atm(argv[f], &ctl, atm);
00076
          00077
00078
00079
08000
00081
00082
             fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00083
          fprintf(out, "\n");
00084
        }
00085
00086
00087
         /* Close file... */
00088
        fclose(out);
00089
00090
        /* Free... */
00091
        free(atm);
00092
00093
        return EXIT_SUCCESS;
00094 }
```

Here is the call graph for this function:



## 5.14 extract.c

```
00001 /*
00002
        This file is part of MPTRAC.
00003
00004
        MPTRAC is free software: you can redistribute it and/or modify
00005
        it under the terms of the GNU General Public License as published by
00006
        the Free Software Foundation, either version 3 of the License, or
00007
        (at your option) any later version.
00008
00009
        MPTRAC is distributed in the hope that it will be useful,
00010
        but WITHOUT ANY WARRANTY; without even the implied warranty of
00011
        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
        GNU General Public License for more details.
00013
        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) 2013-2018 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028
        int argc,
00029
        char *argv[]) {
00030
00031
        ctl_t ctl;
00032
00033
        atm_t *atm;
00034
00035
        FILE *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
00046
        /* Read control parameters... */
00047
         read_ctl(argv[1], argc, argv, &ctl);
00048
        ip = (int) scan_ctl(argv[1], argc, argv, "EXTRACT_IP", -1, "0", NULL);
00049
        /* Write info... */ printf("Write trajectory data: s^n, argv[2]);
00050
00051
00052
00053
         /* Create output file... */
        if (!(out = fopen(argv[2], "w")))
00054
          ERRMSG("Cannot create file!");
00055
00056
00057
         /* Write header... */
00058
        fprintf(out,
00059
                 "# $1 = time [s] \n"
```

```
"# $2 = altitude [km] \n"
00061
                 "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
        for (iq = 0; iq < ctl.nq; iq++)

fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00062
00063
00064
        ctl.qnt_unit[iq]);
fprintf(out, "\n");
00065
00066
00067
        /* Loop over files... */
00068
        for (f = 3; f < argc; f++) {</pre>
00069
          /* Read atmopheric data... */
if (!(in = fopen(argv[f], "r")))
00070
00071
00072
             continue;
00073
00074
             fclose(in);
00075
           read_atm(argv[f], &ctl, atm);
00076
00077
          /* Write data... */
fprintf(out, "%.2f %g %g %g", atm->time[ip],
00078
                    Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
00079
           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
00090
        /* Free... */
00091
        free(atm);
00092
00093
        return EXIT_SUCCESS;
00094 }
```

# 5.15 init.c File Reference

Create atmospheric data file with initial air parcel positions.

#### **Functions**

• int main (int argc, char \*argv[])

#### 5.15.1 Detailed Description

Create atmospheric data file with initial air parcel positions.

Definition in file init.c.

# 5.15.2 Function Documentation

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

Definition at line 27 of file 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_ctl(argv[1], argc, argv, &ctl);

t0 = scan_ctl(argv[1], argc, argv, "INIT_TO", -1, "0", NULL);

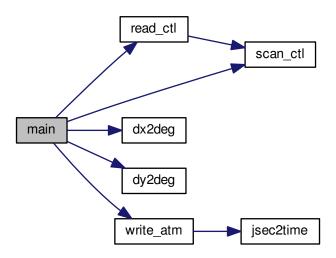
t1 = scan_ctl(argv[1], argc, argv, "INIT_T1", -1, "0", NULL);

dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
00050
00051
00052
             t1 = scan_ctl(argv[1], argc, argv, "INIT_TI", -1, "0", NULL);
dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
z0 = scan_ctl(argv[1], argc, argv, "INIT_ZO", -1, "0", NULL);
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_LONO", -1, "0", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "INIT_LONO", -1, "0", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "INIT_DLON", -1, "1", 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_SINT, -1, "0", NULL);
sz = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
slat = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
sx = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_SX", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_UZ", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_UZ", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_EVENLY", -1, "1", NULL);
rep = (int) scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
00053
00054
00055
00056
00057
00058
00059
00060
00061
00062
00063
00064
00065
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);
00080
               /* Create grid... */
00081
               for (t = t0; t <= t1; t += dt)</pre>
                  for (z = z0; z <= z1; z += dz)
for (lon = lon0; lon <= lon1; lon += dlon)
00082
00083
                           for (lat = lat0; lat <= lat1; lat += dlat)
  for (irep = 0; irep < rep; irep++) {</pre>
00084
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]
                                       = (lon + gsl_ran_gaussian_ziggurat(rng, slon / 2.3548)
00095
                                            + gsl_ran_gaussian_ziggurat(rng, dx2deg(sx, lat) / 2.3548)
+ ulon * (gsl_rng_uniform(rng) - 0.5));
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)
+ ulat * (gsl_rng_uniform(rng) - 0.5));
00101
00102
                                   } while (even && gsl_rng_uniform(rng) >
00103
                                                    fabs(cos(atm->lat[atm->np] * M_PI / 180.)));
00104
00105
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... */
```

5.16 init.c 51

```
if (ctl.qnt_m >= 0)
00117
         for (ip = 0; ip < atm->np; ip++)
00118
            atm->q[ctl.qnt_m][ip] = m / atm->np;
00119
00120
       /* Save data... */
write_atm(argv[2], &ctl, atm, t0);
00121
00122
00123
        /* Free... */
00124
        gsl_rng_free(rng);
00125
        free(atm);
00126
        return EXIT_SUCCESS;
00127
00128 }
```

Here is the call graph for this function:



## 5.16 init.c

```
00001 /*
00002
        This file is part of MPTRAC.
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) 2013-2018 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028
        int argc,
00029
        char *argv[]) {
00030
00031
        atm t *atm;
00032
00033
        ctl_t ctl;
```

```
00034
00035
                qsl rnq *rnq;
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
                 /* Allocate... */
00042
00043
                ALLOC(atm, atm_t, 1);
00044
00045
                /* Check arguments... */
                if (argc < 3)
00046
00047
                    ERRMSG("Give parameters: <ctl> <atm_out>");
00048
00049
                /* 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);
00050
00051
              tl = scan_ctl(argv[1], argc, argv, "INIT_TO", -1, "0", NULL);

tl = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "0", NULL);

tl = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);

z0 = scan_ctl(argv[1], argc, argv, "INIT_ZO", -1, "0", NULL);

z1 = scan_ctl(argv[1], argc, argv, "INIT_ZT", -1, "0", NULL);

dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);

lon0 = scan_ctl(argv[1], argc, argv, "INIT_LONO", -1, "0", NULL);

lon1 = scan_ctl(argv[1], argc, argv, "INIT_LONO", -1, "0", NULL);

dlon = scan_ctl(argv[1], argc, argv, "INIT_LONO", -1, "0", NULL);

lat0 = scan_ctl(argv[1], argc, argv, "INIT_LATO", -1, "0", NULL);

lat1 = scan_ctl(argv[1], argc, argv, "INIT_LATO", -1, "0", NULL);

st = scan_ctl(argv[1], argc, argv, "INIT_DLAT", -1, "1", NULL);

sz = scan_ctl(argv[1], argc, argv, "INIT_ST", -1, "0", NULL);

sz = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);

stat = scan_ctl(argv[1], argc, argv, "INIT_SLATO", -1, "0", NULL);

sx = scan_ctl(argv[1], argc, argv, "INIT_SLATO", -1, "0", NULL);

sx = scan_ctl(argv[1], argc, argv, "INIT_SLATO", -1, "0", NULL);

ut = scan_ctl(argv[1], argc, argv, "INIT_SLATO", -1, "0", NULL);

ut = scan_ctl(argv[1], argc, argv, "INIT_UT, -1, "0", NULL);

ut = scan_ctl(argv[1], argc, argv, "INIT_UT, -1, "0", NULL);

ut = scan_ctl(argv[1], argc, argv, "INIT_UT, -1, "0", NULL);

ut = scan_ctl(argv[1], argc, argv, "INIT_UT, -1, "0", NULL);

ut = scan_ctl(argv[1], argc, argv, "INIT_UT, -1, "0", NULL);

ut = scan_ctl(argv[1], argc, argv, "INIT_UT, -1, "0", NULL);

ut = scan_ctl(argv[1], argc, argv, "INIT_UT, -1, "0", NULL);

ut = scan_ctl(argv[1], argc, argv, "INIT_UT, -1, "0", NULL);

ut = scan_ctl(argv[1], argc, argv, "INIT_UT, -1, "0", NULL);

ut = scan_ctl(argv[1], argc, argv, "INIT_UT, -1, "0", NULL);

ut = scan_ctl(argv[1], argc, argv, "INIT_UT, -1, "0", NULL);

ut = scan_ctl(argv[1], argc, argv, "INIT_UT, -1, "0", NULL);

ut = scan_ctl(argv[1], argc, argv, "INIT_UT, -1, "0", NULL);

ut = scan_ctl(argv[1], argc, argv, "INIT_UT, -1, "0", NULL);

ut = s
00053
00054
00055
00056
00057
00058
00059
00060
00061
00062
00063
00064
00065
00066
00067
00068
00069
00070
                even = (int) scan_ctl(argy[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 <= t1; t += dt)</pre>
                    for (z = z0; z \le z1; z += dz)
00082
                        for (lon = lon0; lon <= lon1; lon += dlon)</pre>
00083
                             for (lat = lat0; lat <= lat1; lat += dlat)</pre>
00085
                                 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)
                                                + 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));
                                     atm->lon[atm->np]
00094
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
                                     do {
00099
                                         atm->lat[atm->np]
                                              = (lat + gsl_ran_gaussian_ziggurat(rng, slat / 2.3548)
00100
00101
                                                    + gsl_ran_gaussian_ziggurat(rng, dy2deg(sx) / 2.3548)
+ ulat * (gsl_rng_uniform(rng) - 0.5));
00102
                                     } while (even && gsl_rng_uniform(rng) >
00104
                                                        fabs(cos(atm->lat[atm->np] * M_PI / 180.)));
00105
                                     /* Set particle counter... */
if ((++atm->np) >= NP)
00106
00107
                                         ERRMSG("Too many particles!");
00108
00109
00110
00111
                 /* Check number of air parcels... */
00112
                if (atm->np <= 0)
                    ERRMSG("Did not create any air parcels!");
00113
00114
                /* Initialize mass... */
00116
                if (ctl.qnt_m >= 0)
00117
                    for (ip = 0; ip < atm->np; ip++)
00118
                        atm->q[ctl.qnt_m][ip] = m / atm->np;
00119
00120
                /* Save data... */
```

```
00121    write_atm(argv[2], &ctl, atm, t0);
00122
00123    /* Free... */
00124    gsl_rng_free(rng);
00125    free(atm);
00126
00127    return EXIT_SUCCESS;
00128 }
```

# 5.17 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
          /\star Check arguments... \star/
          if (argc < 2)
    ERRMSG("Give parameters: <jsec>");
00036
00037
00038
00039
          /* Read arguments... */
00040
         jsec = atof(argv[1]);
00041
00042
         /* Convert time... */
         jsec2time(jsec, &year, &mon, &day, &hour, &min, &sec, &remain);
printf("%d %d %d %d %d %d %g\n", year, mon, day, hour, min, sec, remain);
00043
00044
00045
00046
          return EXIT_SUCCESS;
```

Here is the call graph for this function:



# 5.18 jsec2time.c

```
00001 /*
00002
        This file is part of MPTRAC.
00003
00004
        MPTRAC is free software: you can redistribute it and/or modify
00005
        it under the terms of the GNU General Public License as published by
00006
        the Free Software Foundation, either version 3 of the License, or
        (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) 2013-2018 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028 int arg
        int argc,
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 %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.

double deg2dx (double dlon, double lat)

Convert degrees to horizontal distance.

double deg2dy (double dlat)

Convert degrees to horizontal distance.

double dp2dz (double dp, double p)

Convert pressure to vertical distance.

• double dx2deg (double dx, double lat)

Convert horizontal distance to degrees.

double dy2deg (double dy)

Convert horizontal distance to degrees.

• double dz2dp (double dz, double p)

Convert vertical distance to pressure.

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 \*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 (double \*xx, int n, double x)

Find array index.

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)

  Write profile data.
- void write\_station (const char \*filename, ctl\_t \*ctl, atm\_t \*atm, double t)
   Write station data.

## 5.19.1 Detailed Description

MPTRAC library definitions.

Definition in file libtrac.c.

5.19.2 Function Documentation

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

Convert Cartesian coordinates to geolocation.

Definition at line 29 of file libtrac.c.

```
00033 {
00034
00035 double radius;
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,
00052
            16934400.00, 19612800.00, 22291200.00,
00053
            24883200.00, 27561600.00, 30153600.00
00054
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
              {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\},
              {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
00111
              {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}
             {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},
00126
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},
                {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},
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}}, {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}, {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
00186
00187
                {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
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
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
                {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},
00202
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}, {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},
00221
00222
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
00255
               {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
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
               {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
          int ilat, ip, isec;
00285
00286
00287
           /* Get seconds since begin of year... */
          sec = fmod(t, 365.25 * 86400.);
00288
00289
00290
           /* Get indices... */
          ilat = locate(lats, 18, lat);
00291
          ip = locate(ps, 10, p);
00292
00293
          isec = locate(secs, 12, sec);
00294
00295
           /* Interpolate...
          00296
00297
00298
```

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.

```
00313
00314
00315
           static double dovs[12]
           = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00317
00318
           static double lats[73]
              = { -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,
00322
00324
               45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
00325
              75.
                    77.5, 80, 82.5, 85, 87.5, 90
00326
00327
00328
           static double tps[12][73]
              = { {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, 128,
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
           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,
00336
00337
            300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5, 150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
00338
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.
00341
00342
00343
             284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
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, 161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
00345
00346
00347
             100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
00348
00349
             99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
00350
             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, 304.3, 304.9, 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,
00351
00352
00353
00355
             195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
00356
             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, 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},
00359
00360
            {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
00361
00362
             260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
00363
             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, 101.1, 101.2, 101.5, 101.9, 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,
00364
00365
00366
             273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
00367
00368
             325.3, 325.8, 325.8},
00369
            {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
             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,
00376
                  308.5, 312.2, 313.1, 313.3},
00377
                {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, 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,
00378
00379
                  111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115,
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, 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, 185.2, 186.3, 186.3, 187.1, 189, 182.1, 187.1, 188.3, 187.1, 189, 182.1, 187.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1, 189.1,
00382
00383
00384
00385
                 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,
00386
00387
00388
                  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
                 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},
00391
00393
                {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
00394
                 183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
00395
                 243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
00396
                 114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
                 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, 276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
00397
00398
00400
00401
                {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
00402
                  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, 111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
00403
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,
00406
00407
                 206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
00408
                 279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
00409
                  305.1},
                241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2, 253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
00410
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,
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, 286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
00414
00415
00416
00417
                {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,
00420
                 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, 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,
00421
00422
00423
00424
                 280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
                 281.7, 281.1, 281.2}
00425
00426
00427
00428
               double doy, p0, p1, pt;
00429
                int imon, ilat:
00431
               /* Get day of year... */
doy = fmod(t / 86400., 365.25);
while (doy < 0)
00432
00433
00434
00435
                   dov += 365.25;
00436
00437
                /* Get indices... */
00438
               imon = locate(doys, 12, doy);
00439
               ilat = locate(lats, 73, lat);
00440
00441
                /* Get tropopause pressure... */
               p0 = LIN(lats[ilat], tps[imon][ilat],
00442
                                  lats[ilat + 1], tps[imon][ilat + 1], lat);
               p1 = LIN(lats[ilat], tps[imon + 1][ilat],
lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
00444
00445
00446
               pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
00447
00448
                /* Return tropopause pressure... */
00449
               return pt;
00450 }
```

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 454 of file libtrac.c.

```
00458
00459
          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 \};
00460
00461
00462
          /* Get day of year... */
if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
00463
00464
            *doy = d01[mon - 1] + day - 1;
00465
00466
          else
00467
             *doy = d0[mon - 1] + day - 1;
00468 }
```

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

Get date from day of year.

Definition at line 472 of file libtrac.c.

```
00476
00478
        int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00479
        int d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00480
        int i:
00481
        /* Get month and day... */
if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
00482
00483
         for (i = 11; i >= 0; i--)
00484
00485
           if (d01[i] <= doy)</pre>
          break;
*mon = i + 1;
00486
00487
00488
          *day = doy - d01[i] + 1;
00489
        } else {
         for (i = 11; i >= 0; i--)
00490
          if (d0[i] <= doy)</pre>
00491
          break;
*mon = i + 1;
00492
00493
          *day = doy - d0[i] + 1;
00494
00495
        }
00496 }
```

5.19.2.6 double deg2dx ( double dlon, double lat )

Convert degrees to horizontal distance.

Definition at line 500 of file libtrac.c.

5.19.2.7 double deg2dy ( double dlat )

Convert degrees to horizontal distance.

Definition at line 509 of file libtrac.c.

```
5.19.2.8 double dp2dz ( double dp, double p )
```

Convert pressure to vertical distance.

Definition at line 517 of file libtrac.c.

```
00519 {
00520
00521 return -dp * H0 / p;
00522 }
```

5.19.2.9 double dx2deg ( double dx, double lat )

Convert horizontal distance to degrees.

Definition at line 526 of file libtrac.c.

```
00528 {
00529
00530    /* Avoid singularity at poles... */
00531    if (lat < -89.999 || lat > 89.999)
00532        return 0;
00533    else
00534        return dx * 180. / (M_PI * RE * cos(lat / 180. * M_PI));
00535 }
```

5.19.2.10 double dy2deg ( double dy )

Convert horizontal distance to degrees.

Definition at line 539 of file libtrac.c.

```
00540 {
00541
00542 return dy * 180. / (M_PI * RE);
00543 }
```

5.19.2.11 double dz2dp ( double dz, double p )

Convert vertical distance to pressure.

Definition at line 547 of file libtrac.c.

```
00549 {
00550
00551 return -dz * p / H0;
00552 }
```

5.19.2.12 void geo2cart (double z, double lon, double lat, double \*x)

Convert geolocation to Cartesian coordinates.

Definition at line 556 of file libtrac.c.

```
00560 {
00561
00562 double radius;
00563
00564 radius = z + RE;
00565 x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
00566 x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
00567 x[2] = radius * sin(lat / 180 * M_PI);
```

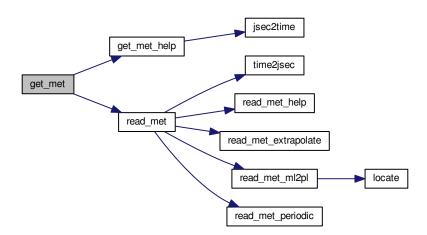
5.19.2.13 void get\_met ( ctl\_t \* ctl, char \* metbase, double t, met\_t \* met0, met\_t \* met1 )

Get meteorological data for given timestep.

Definition at line 572 of file libtrac.c.

```
00577
00578
        static int init;
00580
00581
        char filename[LEN];
00582
00583
        /* Init... */
        if (t == ctl->t_start || !init) {
  init = 1;
00584
00585
00586
00587
           get_met_help(t, -1, metbase, ctl->dt_met, filename);
           read_met(ctl, filename, met0);
00588
00589
          get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
00590
      dt_met, filename);
00591
          read_met(ctl, filename, metl);
00592
00593
        /\star Read new data for forward trajectories... \star/
00594
00595
        if (t > met1->time && ct1->direction == 1) {
         memcpy(met0, met1, sizeof(met_t));
00596
          get_met_help(t, 1, metbase, ctl->dt_met, filename);
read_met(ctl, filename, met1);
00598
00599
00600
00601
        /* Read new data for backward trajectories... */
        if (t < met0->time && ctl->direction == -1) {
00602
         memcpy(met1, met0, sizeof(met_t));
          get_met_help(t, -1, metbase, ctl->dt_met, filename);
read_met(ctl, filename, met0);
00604
00605
00606
00607 }
```

Here is the call graph for this function:



5.19.2.14 void get\_met\_help ( double t, int direct, char \* metbase, double dt\_met, char \* filename )

Get meteorological data for timestep.

Definition at line 611 of file libtrac.c.

```
{
00618
        double t6, r;
00619
00620
        int year, mon, day, hour, min, sec;
00621
00622
        /\star Round time to fixed intervals... \star/
00623
        if (direct == -1)
00624
          t6 = floor(t / dt_met) * dt_met;
00625
        else
00626
          t6 = ceil(t / dt_met) * dt_met;
00627
00628
        /* Decode time... */
00629
        jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00630
        /* Set filename... */
sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", metbase, year, mon, day, hour);
00631
00632
00633 }
```

Here is the call graph for this function:



5.19.2.15 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 637 of file libtrac.c.

```
00643
00644
00645
          double aux00, aux01, aux10, aux11;
00646
00647
          /* Set variables...
00648 aux00 = array[ix][iy];

00649 aux01 = array[ix][iy + 1];

00650 aux10 = array[ix + 1][iy];
00651
          aux11 = array[ix + 1][iy + 1];
00652
00653
          /* Interpolate horizontally... */
         aux00 = wy * (aux00 - aux01) + aux01;
aux11 = wy * (aux10 - aux11) + aux11;
00654
00656
          *var = wx * (aux00 - aux11) + aux11;
00657 }
```

5.19.2.16 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 661 of file libtrac.c.

```
00670
00671
          double aux00, aux01, aux10, aux11;
00672
00673
         /* Interpolate vertically... */
00674
         aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
            + array[ix][iy][ip + 1];
00676
         aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
         tartay[ix][iy + 1][ip + 1];
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])
00677
00678
00679
00680
          + array[ix + 1][iy + 1][ip + 1];
00681
00682
00683
         /* Interpolate horizontally... */
         aux00 = wy * (aux00 - aux01) + aux01;

aux11 = wy * (aux10 - aux11) + aux11;
00684
00685
         *var = wx * (aux00 - aux11) + aux11;
00686
00687 }
```

5.19.2.17 void intpol\_met\_space (  $met_t* met$ , double p, double lon, double lon

Spatial interpolation of meteorological data.

Definition at line 691 of file libtrac.c.

```
00705
00706
00707
        double wp, wx, wy;
00708
00709
        int ip, ix, iy;
00710
        /* Check longitude... */
if (met->lon[met->nx - 1] > 180 && lon < 0)
00711
00712
00713
          lon += 360;
00714
        /* Get indices... */
00715
00716
        ip = locate(met->p, met->np, p);
        ix = locate(met->lon, met->nx, lon);
iy = locate(met->lat, met->ny, lat);
00717
00718
00719
00720
00721
        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]);
00722
00723
00724
00725
        /* Interpolate... */
00726
        if (ps != NULL)
00727
          intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
00728
        if (pt != NULL)
00729
          intpol_met_2d(met->pt, ix, iy, wx, wy, pt);
00730
        if (z != NULL)
          intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy, z);
00732
        if (t != NULL)
00733
          intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
00734
        if (u != NULL)
00735
          intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
        if (v != NULL)
00736
00737
          intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
00738
        if (w != NULL)
00739
           intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
00740
        if (pv != NULL)
00741
          intpol_met_3d(met->pv, ip, ix, iy, wp, wx, wy, pv);
00742
        if (h2o != NULL)
          intpol_met_3d (met->h2o, ip, ix, iy, wp, wx, wy, h2o);
00743
00744
        if (o3 != NULL)
00745
          intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00746 }
```

Here is the call graph for this function:

5.19.2.18 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 750 of file libtrac.c.

```
00766
00768
       double h200, h201, o30, o31, ps0, ps1, pt0, pt1, pv0, pv1, t0, t1, u0, u1,
00769
          v0, v1, w0, w1, wt, z0, z1;
00770
00771
        /* Spatial interpolation...
00772
       00773
00774
                         pt == NULL ? NULL : &pt0,
00775
                         z == NULL ? NULL : &z0,
                         t == NULL ? NULL : &t0,
00776
00777
                         u == NULL ? NULL : &u0,
00778
                         v == NULL ? NULL : &v0,
00779
                         w == NULL ? NULL : &w0,
00780
                         pv == NULL ? NULL : &pv0,
00781
                         h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00782
        intpol_met_space(met1, p, lon, lat,
00783
                         ps == NULL ? NULL : &ps1,
                         pt == NULL ? NULL : &pt1,
00784
00785
                         z == NULL ? NULL : &z1,
00786
                         t == NULL ? NULL : &t1,
00787
                         u == NULL ? NULL : &u1,
                         v == NULL ? NULL : &v1,
00788
00789
                         w == NULL ? NULL : &w1,
                         pv == NULL ? NULL : &pv1,
h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00790
00791
00792
00793
       /* Get weighting factor... */
00794
       wt = (met1->time - ts) / (met1->time - met0->time);
00795
00796
       /* Interpolate... */
       if (ps != NULL)
00797
00798
          *ps = wt * (ps0 - ps1) + ps1;
00799
       if (pt != NULL)
       *pt = wt * (pt0 - pt1) + pt1;
if (z != NULL)
00800
00801
00802
         *z = wt * (z0 - z1) + z1;
       if (t != NULL)
00803
00804
         *t = wt * (t0 - t1) + t1;
00805
       if (u != NULL)
00806
          *u = wt * (u0 - u1) + u1;
       if (v != NULL)
00807
         *v = wt * (v0 - v1) + v1;
80800
       if (w != NULL)

*w = wt * (w0 - w1) + w1;
00809
00810
       if (pv != NULL)
00811
00812
          \star pv = wt \star (pv0 - pv1) + pv1;
00813
       if (h2o != NULL)
00814
         *h2o = wt * (h2o0 - h2o1) + h2o1;
       if (o3 != NULL)
00815
          *o3 = wt * (o30 - o31) + o31;
00816
00817 }
```

Here is the call graph for this function:

```
5.19.2.19 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 821 of file libtrac.c.

```
00829 {
00830
00831 struct tm t0, *t1;
00832
00833 time_t jsec0;
00834
00835 t0.tm_year = 100;
```

```
t0.tm_mon = 0;
         t0.tm_mday = 1;
t0.tm_hour = 0;
00838
         t0.tm_min = 0;
00839
         t0.tm_sec = 0;
00840
00841
00842
         jsec0 = (time_t) jsec + timegm(&t0);
00843
         t1 = gmtime(&jsec0);
00844
         *year = t1->tm_year + 1900;
00845
        *mon = t1->tm_year + ;

*day = t1->tm_mon + 1;
00846
00847
         *hour = t1->tm_hour;

*min = t1->tm_min;
00848
00849
00850
         *sec = t1->tm_sec;
00851
        *remain = jsec - floor(jsec);
00852 }
```

## 5.19.2.20 int locate ( double \*xx, int n, double x )

Find array index.

Definition at line 856 of file libtrac.c.

```
00859
                    {
00860
00861
         int i, ilo, ihi;
00863
        ilo = 0;
00864
        ihi = n - 1;
        i = (ihi + ilo) >> 1;
00865
00866
00867
         if (xx[i] < xx[i + 1])
          while (ihi > ilo + 1) {
  i = (ihi + ilo) >> 1;
00869
00870
             if (xx[i] > x)
00871
               ihi = i;
00872
             else
              ilo = i;
00873
00874
        } else
00875
         while (ihi > ilo + 1) {
            i = (ihi + ilo) >> 1;
00876
             if (xx[i] <= x)
  ihi = i;</pre>
00877
00878
00879
             else
              ilo = i;
00880
00881
00882
00883
        return ilo;
00884 }
```

5.19.2.21 void read\_atm ( const char \* filename, ctl\_t \* ctl, atm\_t \* atm )

Read atmospheric data.

Definition at line 888 of file libtrac.c.

```
00891
00892
00893
       FILE *in;
00894
00895
       char line[LEN], *tok;
00896
00897
       double t0:
00898
00899
       int dimid, ip, iq, ncid, varid;
00900
00901
       size_t nparts;
00902
00903
       /* Init... */
00904
       atm->np = 0;
00905
00906
       /* Write info... */
       printf("Read atmospheric data: %s\n", filename);
```

```
00908
00909
          /* Read ASCII data... */
00910
          if (ctl->atm_type == 0) {
00911
00912
             /* Open file... */
00913
            if (!(in = fopen(filename, "r")))
               ERRMSG("Cannot open file!");
00915
00916
             /* Read line... */
00917
            while (fgets(line, LEN, in)) {
00918
               /* 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]);
00919
00920
00921
00922
00923
00924
00925
00926
00927
               /* Convert altitude to pressure... */
00928
               atm \rightarrow p[atm \rightarrow np] = P(atm \rightarrow p[atm \rightarrow np]);
00929
               /* Increment data point counter... */
if ((++atm->np) > NP)
00930
00931
00932
                  ERRMSG("Too many data points!");
00933
00934
00935
             /* Close file... */
00936
            fclose(in);
00937
00938
00939
          /* Read binary data... */
00940
          else if (ctl->atm_type == 1) {
00941
00942
             /* Open file... */
            if (!(in = fopen(filename, "r")))
00943
               ERRMSG("Cannot open file!");
00944
00946
             /* Read data... */
00947
            FREAD(&atm->np, int,
00948
                    1.
00949
                    in):
            FREAD (atm->time, double,
00950
00951
                      (size_t) atm->np,
                    in);
00952
00953
            FREAD(atm->p, double,
00954
                      (size_t) atm->np,
00955
                    in);
00956
            FREAD (atm->lon, double,
00957
                      (size t) atm->np.
00958
                    in);
00959
            FREAD(atm->lat, double,
00960
                      (size_t) atm->np,
            in);
for (iq = 0; iq < ctl->nq; iq++)
FREAD(atm->q[iq], double,
00961
00962
00963
00964
                         (size_t) atm->np,
00965
00966
            /* Close file... */
00967
00968
            fclose(in);
00969
00970
00971
          /* Read netCDF data... */
00972
          else if (ctl->atm_type == 2) {
00973
00974
             /* Open file... */
            NC(nc_open(filename, NC_NOWRITE, &ncid));
00975
00976
               Get dimensions... */
00978
            NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
00979
            NC(nc_inq_dimlen(ncid, dimid, &nparts));
            atm->np = (int) nparts;
if (atm->np > NP)
00980
00981
               ERRMSG("Too many particles!");
00982
00983
00984
00985
            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;
00986
00987
00988
            /* Read geolocations... */
NC(nc_inq_varid(ncid, "PRESS", &varid));
00990
00991
            NC(nc_get_var_double(ncid, varid, atm->p));
NC(nc_inq_varid(ncid, "LON", &varid));
00992
00993
00994
            NC(nc_get_var_double(ncid, varid, atm->lon));
```

```
NC(nc_inq_varid(ncid, "LAT", &varid));
00996
           NC(nc_get_var_double(ncid, varid, atm->lat));
00997
00998
            /* Read variables... */
00999
           if (ctl->qnt_p >= 0)
             if (nc_ing_varid(ncid, "PRESS", &varid) == NC_NOERR)
01000
                NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
01002
           if (ctl->qnt_t >= 0)
01003
             if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
01004
                NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
01005
           if (ctl->qnt_u >= 0)
             if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
01006
           NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
if (ctl->qnt_v >= 0)
01007
01008
01009
             if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
01010
                \label{eq:ncd_def} \mbox{NC(nc\_get\_var\_double(ncid, varid, atm->q[ctl->qnt\_v]));}
01011
           if (ctl->qnt_w >= 0)
             if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
01012
               NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
           if (ctl->qnt_h2o >= 0)
01014
             if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
01015
01016
01017
           if (ctl->qnt_o3 >= 0)
           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]));
if (ctl->qnt_theta >= 0)
  if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
01018
01019
01020
01021
01022
               NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
01023
           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]));
01024
01025
01026
01027
           for (ip = 0; ip < atm->np; ip++)
01028
             01029
01030
01031
01032
01033
                   || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
01034
                atm->time[ip] = GSL_NAN;
                atm->p[ip] = GSL_NAN;
atm->lon[ip] = GSL_NAN;
01035
01036
                atm->lat[ip] = GSL_NAN;
for (iq = 0; iq < ctl->nq; iq++)
01037
01038
                 atm->q[iq][ip] = GSL_NAN;
01039
01040
             } else {
01041
               if (ctl->qnt_h2o >= 0)
01042
                  atm->q[ctl->qnt_h2o][ip] *= 1.608;
                if (ctl->qnt_pv >= 0)
01043
                atm->q[ctl->qnt_pv][ip] *= 1e6;
if (atm->lon[ip] > 180)
01044
01045
01046
                  atm->lon[ip] -= 360;
01047
01048
            /* Close file... */
01049
01050
           NC(nc close(ncid));
01051
01052
01053
         /* Error... */
01054
           ERRMSG("Atmospheric data type not supported!");
01055
01056
01057
         /* Check number of points... */
01058
         if (atm->np < 1)
01059
           ERRMSG("Can not read any data!");
01060 }
```

5.19.2.22 void read\_ctl ( const char \* filename, int argc, char \* argv[], ctl\_t \* ctl )

Read control parameters.

Definition at line 1064 of file libtrac.c.

```
01068 {
01069
01070 int ip, iq;
01071
01072 /* Write info... */
01073 printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
01074 "(executable: %s | compiled: %s, %s)\n\n",
```

```
argv[0], __DATE__, __TIME__);
01076
01077
        /* Initialize quantity indices... */
01078
        ctl->qnt_ens = -1;
        ctl->qnt_m = -1;
01079
        ctl->qnt_r = -1;
01080
        ctl->qnt_rho = -1;
        ctl->qnt_ps = -1;
01082
01083
        ctl->qnt_pt = -1;
        ctl->qnt_z = -1;
01084
        ct1->qnt_p = -1;
01085
        ctl->qnt_t = -1;
01086
        ctl->qnt_u = -1;
01087
        ctl->qnt_v = -1;
01088
01089
        ctl->qnt_w = -1;
01090
        ctl->qnt_h2o = -1;
        ctl->qnt_o3 = -1;
01091
01092
        ctl->qnt theta = -1;
01093
        ctl \rightarrow qnt_vh = -1;
01094
        ctl->qnt_vz = -1;
01095
        ctl->qnt_pv = -1;
01096
        ctl->qnt\_tice = -1;
        ctl->qnt\_tsts = -1;
01097
        ctl->qnt\_tnat = -1;
01098
01099
        ctl \rightarrow qnt_stat = -1;
01100
01101
        /* Read quantities... */
01102
        ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
01103
        if (ctl->nq > NQ)
         ERRMSG("Too many quantities!");
01104
01105
        for (iq = 0; iq < ctl->nq; iq++) {
01106
01107
           /* 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",
01108
01109
01110
                    ctl->qnt_format[iq]);
01111
          /* Try to identify quantity... */
if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
01112
01113
           ctl->qnt_ens = iq;
01114
            sprintf(ctl->qnt_unit[iq], "-");
01115
          } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
01116
            ctl->qnt_m = iq;
01117
01118
            sprintf(ctl->qnt_unit[iq], "kg");
          } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
01119
            ctl->qnt_r = iq;
01120
01121
            sprintf(ctl->qnt_unit[iq], "m");
          } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
ctl->qnt_rho = iq;
01122
01123
01124
            sprintf(ctl->qnt_unit[iq], "kg/m^3");
01125
          } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
01126
           ctl->qnt_ps = iq;
01127
            sprintf(ctl->qnt_unit[iq], "hPa");
01128
          } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
            ctl->qnt_pt = iq;
01129
            sprintf(ctl->qnt_unit[iq], "hPa");
01130
          } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
01131
            ctl->qnt_z = iq;
01132
01133
            sprintf(ctl->qnt_unit[iq], "km");
          } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
ctl->qnt_p = iq;
01134
01135
            sprintf(ctl->qnt_unit[iq], "hPa");
01136
          } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
01137
           ctl->qnt_t = iq;
01138
01139
            sprintf(ctl->qnt_unit[iq], "K");
01140
          } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
01141
            ctl->qnt_u = iq;
            sprintf(ctl->qnt_unit[iq], "m/s");
01142
01143
          } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
01144
            ctl->qnt_v = iq;
01145
            sprintf(ctl->qnt_unit[iq], "m/s");
01146
          else\ if\ (strcmp(ctl->qnt_name[iq], "w") == 0) {
            ctl->qnt_w = iq;
sprintf(ctl->qnt_unit[iq], "hPa/s");
01147
01148
          } else if (strcmp(ct1->qnt_name[iq], "h2o") == 0) {
  ct1->qnt_h2o = iq;
01149
01150
01151
            sprintf(ctl->qnt_unit[iq], "1");
01152
          } else if (strcmp(ct1->qnt_name[iq], "o3") == 0) {
01153
            ct1->qnt_o3 = iq;
            sprintf(ctl->qnt_unit[iq], "1");
01154
          } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
01155
           ctl->qnt_theta = iq;
sprintf(ctl->qnt_unit[iq], "K");
01156
01157
01158
          } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
            ctl->qnt_vh = iq;
01159
            sprintf(ctl->qnt_unit[iq], "m/s");
01160
01161
          } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
```

```
01162
             ctl->qnt_vz = iq;
             sprintf(ctl->qnt_unit[iq], "m/s");
01163
           } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
01164
              ctl->qnt_pv = iq;
01165
01166
             sprintf(ctl->qnt_unit[iq], "PVU");
           } else if (stromp(ctl->qnt_name[iq], "tice") == 0) {
  ctl->qnt_tice = iq;
01167
01168
01169
             sprintf(ctl->qnt_unit[iq], "K");
01170
           } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
             ctl->qnt_tsts = iq;
01171
             sprintf(ctl->qnt_unit[iq], "K");
01172
           } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
01173
             ctl->qnt_tnat = iq;
01174
             sprintf(ctl->qnt_unit[iq], "K");
01175
01176
           } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
              ctl->qnt_stat = iq;
01177
             sprintf(ctl->qnt_unit[iq], "-");
01178
01179
           } else
01180
             scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
01181
01182
01183
         /\star Time steps of simulation... \star/
01184
         ctl->direction =
           (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
f (ctl->direction != -1 && ctl->direction != 1)
01185
01186
           ERRMSG("Set DIRECTION to -1 or 1!");
01187
        ctl->t_stop = scan_ctl(filename, argo, argv, "T_STOP", -1, "1e100", NULL);
ctl->dt_mod = scan_ctl(filename, argo, argv, "DT_MOD", -1, "600", NULL);
01188
01189
01190
01191
         /* Meteorological data... */
         ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
01192
         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);
01193
01194
01195
         ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
         ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -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);
01196
01197
01198
         ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
01199
01200
         if
            (ctl->met_np > EP)
          ERRMSG("Too many levels!");
01201
01202
         for (ip = 0; ip < ctl->met_np; ip++)
          ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
01203
01204
         ctl->met tropo
01205
           = (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);
01206
01207
01208
01209
         /\star Isosurface parameters... \star/
01210
         ctl->isosurf
         = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
01211
01212
01213
01214
         /* Diffusion parameters... */
01215
         ctl->turb_dx_trop
           = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
01216
01217
         ctl->turb dx strat
01218
            = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
01219
         ctl->turb dz trop
01220
           = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
01221
         ctl->turb_dz_strat
01222
           = scan ctl(filename, argc, argv, "TURB DZ STRAT", -1, "0.1", NULL);
01223
         ctl->turb mesox =
01224
           scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
01225
01226
           scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
01227
01228
        /* Mass and life time...
         ctl->molmass = scan_ctl(filename, argc, argv, "MOLMASS", -1, "1", NULL);
01229
         ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
01230
01231
         ctl->tdec strat =
01232
           scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
01233
01234
         /* PSC analysis... */
        ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL); ctl->psc_hno3 =
01235
01236
01237
           scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
01238
         /* Output of atmospheric data... */
scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
01239
01240
      atm basename):
01241
         scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
01242
         ctl->atm_dt_out
01243
           scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
01244
         ctl->atm_filter
01245
           (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
         ctl->atm_type =
01246
01247
           (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
```

```
01248
01249
          /* Output of CSI data... */
         scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
01250
       csi basename);
01251 ctl->csi dt out =
         scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->
01252
01253
       csi_obsfile);
01254 ctl->csi_obsmin =
01255
           scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
01256
         ctl->csi modmin =
           scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
01257
         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);
01258
01259
01260
01261
         ctl->csi_lon0 =
         scan_ctl(filename, argc, argv, "CSI_LONO", -1, "-180", NULL);
ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1,
01262
                                                                                       "180", NULL);
01263
01264
         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);
01265
01266
01267
01268
         ctl->csi nv =
            (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
01269
01270
01271
         /* Output of ensemble data... */
01272
         scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
       ens_basename);
01273
01274
         /* Output of grid data... */
         scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
01275
                    ctl->grid_basename);
         scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
01277
       grid_gpfile);
01278 ctl->grid_dt_out =
            scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
01279
01280
         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);
01282
01283
         ctl->grid_nz =
01284
            (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
01285
01286
         ct1->grid lon0 =
01287
           scan_ctl(filename, argc, argv, "GRID_LONO", -1, "-180", NULL);
01288
         ctl->grid lon1 =
01289
            scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
         ctl->grid_nx =
01290
           (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
01291
01292
         ctl->grid lat0 =
01293
            scan_ctl(filename, argc, argv, "GRID_LATO", -1, "-90", NULL);
01294
         ctl->grid_lat1
01295
            scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
01296
01297
           (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
01298
01299
         /* Output of profile data... */
         scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
01300
01301
                     ctl->prof_basename);
         scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
01302
prof_obsfile);
01303 ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
01304 ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
01305
         ctl->prof_nz =
01306
            (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
01307
         ctl->prof_lon0 =
01308
           scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
01309
         ctl->prof lon1
            scan ctl(filename, argc, argv, "PROF LON1", -1, "180", NULL);
01310
01311
         ctl->prof nx =
01312
            (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
01313
          ctl->prof_lat0 =
01314
           scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
01315
         ctl->prof_lat1 =
            scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
01316
         ctl->prof nv :
01317
01318
            (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
01319
01320
         /* Output of station data... */
         scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
01321
01322
                    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);
01323
01324
01325
01326 }
```

Here is the call graph for this function:



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

Read meteorological data file.

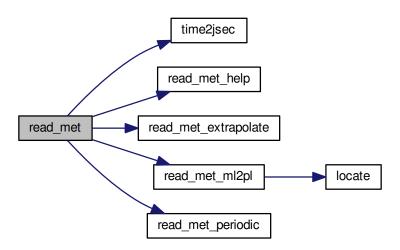
Definition at line 1330 of file libtrac.c.

```
01333
01334
01335
        char cmd[2 * LEN], levname[LEN], tstr[10];
01336
01337
        static float help[EX * EY];
01338
        int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
01340
01341
        size_t np, nx, ny;
01342
        /* Write info... */
01343
01344
        printf("Read meteorological data: %s\n", filename);
01345
01346
         /* Get time from filename... */
01347
        sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
        year = atoi(tstr);
sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
01348
01349
01350
        mon = atoi(tstr);
         sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
01351
01352
        day = atoi(tstr);
01353
         sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
01354
        hour = atoi(tstr);
01355
        time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
01356
        /* Open netCDF file... */
if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
01357
01358
01359
01360
           /\star Try to stage meteo file...
           if (ctl->met_stage[0] != '-') {
    sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
01361
01362
             year, mon, day, hour, filename);
if (system(cmd) != 0)
01363
01364
01365
               ERRMSG("Error while staging meteo data!");
01366
01367
           /* Try to open again... */
01368
           NC(nc_open(filename, NC_NOWRITE, &ncid));
01369
01371
         /* Get dimensions... */
01372
        NC(nc_inq_dimid(ncid, "lon", &dimid));
NC(nc_inq_dimlen(ncid, dimid, &nx));
if (nx < 2 || nx > EX)
01373
01374
01375
01376
           ERRMSG("Number of longitudes out of range!");
01377
01378
        NC(nc_inq_dimid(ncid, "lat", &dimid));
01379
        NC(nc_inq_dimlen(ncid, dimid, &ny));
        if (ny < 2 || ny > EY)
    ERRMSG("Number of latitudes out of range!");
01380
01381
01382
        sprintf(levname, "lev");
01383
01384
         NC(nc_inq_dimid(ncid, levname, &dimid));
01385
        NC(nc_inq_dimlen(ncid, dimid, &np));
01386
         if (np == 1) {
01387
           sprintf(levname, "lev_2");
01388
           NC(nc_inq_dimid(ncid, levname, &dimid));
01389
           NC(nc_inq_dimlen(ncid, dimid, &np));
```

```
01390
         if (np < 2 || np > EP)
01391
01392
           ERRMSG("Number of levels out of range!");
01393
01394
         /* Store dimensions... */
         met->np = (int) np;
met->nx = (int) nx;
01395
01396
01397
         met->ny = (int) ny;
01398
         /* Get horizontal grid... */
NC(nc_inq_varid(ncid, "lon", &varid));
01399
01400
         NC(nc_get_var_double(ncid, varid, met->lon));
NC(nc_inq_varid(ncid, "lat", &varid));
01401
01402
01403
         NC(nc_get_var_double(ncid, varid, met->lat));
01404
         /* 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);
01405
01406
01407
01408
         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));
01409
01410
01411
01412
01413
         /* Meteo data on pressure levels... */
01414
         if (ctl->met_np <= 0) {</pre>
01415
01416
            /* Read pressure levels from file...
01417
           NC(nc_inq_varid(ncid, levname, &varid));
01418
           NC(nc_get_var_double(ncid, varid, met->p));
01419
           for (ip = 0; ip < met->np; ip++)
             met->p[ip] /= 100.;
01420
01421
01422
            /* Extrapolate data for lower boundary... */
01423
           read_met_extrapolate(met);
01424
01425
01426
         /* Meteo data on model levels... */
01427
         else {
01428
           /* Read pressure data from file... */
read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
01429
01430
01431
            /\star Interpolate from model levels to pressure levels... \star/
01432
01433
           read_met_ml2pl(ctl, met, met->t);
            read_met_ml2pl(ctl, met, met->u);
01434
            read_met_ml2p1(ctl, met, met->v);
01435
01436
            read_met_ml2pl(ctl, met, met->w);
01437
            read_met_ml2pl(ctl, met, met->h2o);
01438
            read_met_ml2pl(ctl, met, met->o3);
01439
01440
            /* Set pressure levels... */
           met->np = ctl->met_np;
for (ip = 0; ip < met->np; ip++)
01441
01442
01443
             met->p[ip] = ctl->met_p[ip];
01444
01445
         /* Check ordering of pressure levels... */
         for (ip = 1; ip < met->np; ip++)
  if (met->p[ip - 1] < met->p[ip])
01447
01448
01449
              ERRMSG("Pressure levels must be descending!");
01450
01451
         01452
01453
01454
           NC(nc_get_var_float(ncid, varid, help));
01455
            for (iy = 0; iy < met->ny; iy++)
              for (ix = 0; ix < met \rightarrow nx; ix++)
01456
         met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
} else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR
01457
01458
                      || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
01459
01460
            NC(nc_get_var_float(ncid, varid, help));
01461
            for (iy = 0; iy < met->ny; iy++)
              for (ix = 0; ix < met->nx; ix++)
01462
                met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
01463
01464
         } else
01465
           for (ix = 0; ix < met->nx; ix++)
              for (iy = 0; iy < met->ny; iy++)
01466
01467
                met->ps[ix][iy] = met->p[0];
01468
01469
         /* Create periodic boundary conditions... */
01470
         read met periodic (met);
01471
01472
         /* Calculate geopotential heights... */
01473
         read_met_geopot(ctl, met);
01474
01475
         /* Calculate potential vorticity... */
01476
         read_met_pv(met);
```

```
01477
01478
        /* Calculate tropopause pressure... */
01479
        read_met_tropo(ctl, met);
01480
01481
       /* Downsampling... */
       read_met_sample(ctl, met);
01482
01483
01484
        /* Close file... */
01485
       NC(nc_close(ncid));
01486 }
```

Here is the call graph for this function:



## 5.19.2.24 void read\_met\_extrapolate ( met\_t \* met )

Extrapolate meteorological data at lower boundary.

Definition at line 1490 of file libtrac.c.

```
01492
01493
            int ip, ip0, ix, iy;
01494
            /* Loop over columns... */
for (ix = 0; ix < met->nx; ix++)
   for (iy = 0; iy < met->ny; iy++) {
01495
01496
01497
01498
01499
                  /* Find lowest valid data point... */
                  for (ip0 = met->np - 1; ip0 >= 0; ip0--)
   if (!gsl_finite(met->t[ix][iy][ip0])
01500
01501
                           || !gsl_finite(met->u[ix][iy][ip0])
|| !gsl_finite(met->v[ix][iy][ip0])
01502
01503
01504
                           || !gsl_finite(met->w[ix][iy][ip0]))
01505
01506
                  /* Extrapolate... */
for (ip = ip0; ip >= 0; ip--) {
  met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
01507
01508
01509
01510
                     met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
                    met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];

met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];

met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
01512
01513
                     met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
01514
01515
01516
               }
01517 }
```

```
5.19.2.25 void read_met_geopot ( ctl_t * ctl, met_t * met )
```

Calculate geopotential heights.

Definition at line 1521 of file libtrac.c.

```
01523
01524
01525
        static double topo_lat[EY], topo_lon[EX], topo_z[EX][EY];
01526
        static int init, topo_nx = -1, topo_ny;
01527
01528
01529
        FILE *in;
01531
        char line[LEN];
01532
01533
        double data[30], lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;
01534
01535
        float help[EX][EY];
01536
        int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;
01538
01539
        /* Initialize geopotential heights... */
01540
        for (ix = 0; ix < met->nx; ix++)
         for (iy = 0; iy < met >ny; iy++)
for (ip = 0; ip < met ->ny; ip++)
01541
01542
01543
              met->z[ix][iy][ip] = GSL_NAN;
01544
01545
        /* Check filename...
        if (ctl->met_geopot[0] == '-')
01546
01547
          return;
01548
        /* Read surface geopotential... */
01550
        if (!init) {
01551
          init = 1;
01552
          /* Write info... */
01553
01554
         printf("Read surface geopotential: %s\n", ctl->met_geopot);
01555
01556
          if (!(in = fopen(ctl->met_geopot, "r")))
    ERRMSG("Cannot open file!");
01557
01558
01559
01560
          /* Read data... */
          while (fgets(line, LEN, in))
if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rz) == 3) {
01561
01562
01563
              if (rlon != rlon_old) {
               if ((++topo_nx) >= EX)
   ERRMSG("Too many longitudes!");
01564
01565
                topo_ny = 0;
01566
01567
01568
              rlon_old = rlon;
01569
              topo_lon[topo_nx] = rlon;
              topo_lat[topo_ny] = rlat;
01570
              topo_z[topo_nx][topo_ny] = rz;
if ((++topo_ny) >= EY)
01571
01572
01573
                ERRMSG("Too many latitudes!");
01574
01575
          if ((++topo_nx) >= EX)
01576
            ERRMSG("Too many longitudes!");
01577
01578
          /* Close file... */
01579
          fclose(in);
01581
          /* Check grid spacing... */
01582
          || fabs(met->lat[0] - met->lat[1]) != fabs(topo_lat[0] - topo_lat[1]))
01583
01584
            printf("Warning: Grid spacing does not match!\n");
01585
01586
01587
        /\star Apply hydrostatic equation to calculate geopotential heights... \star/
01588
        for (ix = 0; ix < met->nx; ix++)
01589
          for (iy = 0; iy < met->ny; iy++) {
01590
            /* Get surface height... */
01591
01592
            lon = met->lon[ix];
01593
            if (lon < topo_lon[0])</pre>
              lon += 360;
01594
            else if (lon > topo_lon[topo_nx - 1])
lon -= 360;
01595
01596
01597
            lat = met->lat[iy];
01598
            tx = locate(topo_lon, topo_nx, lon);
01599
            ty = locate(topo_lat, topo_ny, lat);
```

```
z0 = LIN(topo_lon[tx], topo_z[tx][ty],
                      topo_lon[tx + 1], topo_z[tx + 1][ty], lon);
01601
            z1 = LIN(topo_lon[tx], topo_z[tx][ty + 1],
topo_lon[tx + 1], topo_z[tx + 1][ty + 1], lon);
01602
01603
            z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);
01604
01605
01606
             /* Find surface pressure level... */
01607
            ip0 = locate(met->p, met->np, met->ps[ix][iy]);
01608
            01609
01610
01611
01612
01613
             /* Upper part of profile... */
01614
            met \rightarrow 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]));
01615
01616
            for (ip = ip0 + 2; ip < met->np; ip++)
01617
              met->z[ix][iy][ip]
01618
01619
                = (float) (met->z[ix][iy][ip - 1] + RI / MA / G0
                             * 0.5 * (met->t[ix][iy][ip - 1] + met->t[ix][iy][ip]) 
* log(met->p[ip - 1] / met->p[ip]));
01620
01621
01622
          }
01623
01624
        /* Smooth fields... */
        for (ip = 0; ip < met->np; ip++) {
01625
01626
           /* Median filter...
01627
01628
          for (ix = 0; ix < met->nx; ix++)
01629
            for (iy = 0; iy < met->nx; iy++) {
01630
              n = 0;
01631
               for (ix2 = ix - 2; ix2 \le ix + 2; ix2++) {
01632
                ix3 = ix2;
01633
                 if (ix3 < 0)
01634
                   ix3 += met->nx;
                 if (ix3 >= met -> nx)
01635
                  ix3 -= met -> nx;
01636
                 for (iy2 = GSL_MAX(iy - 2, 0); iy2 <= GSL_MIN(iy + 2, met->ny - 1);
01637
                      iy2++)
01638
01639
                   if (gsl_finite(met->z[ix3][iy2][ip])) {
01640
                     data[n] = met \rightarrow z[ix3][iy2][ip];
01641
                    n++;
                  }
01642
01643
               if (n > 0) {
01644
01645
                 gsl_sort(data, 1, (size_t) n);
01646
                 help[ix][iy] = (float)
01647
                   gsl_stats_median_from_sorted_data(data, 1, (size_t) n);
              } else
01648
01649
                 help[ix][iy] = GSL_NAN;
01650
            }
01651
01652
          /* Copy data... */
          for (ix = 0; ix < met->nx; ix++)
  for (iy = 0; iy < met->nx; iy++)
01653
01654
              met \rightarrow z[ix][iy][ip] = help[ix][iy];
01655
01656
01657 }
```

Here is the call graph for this function:

5.19.2.26 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 1661 of file libtrac.c.

```
01667
                    {
01668
       static float help[EX * EY * EP];
01669
01670
01671
        int ip, ix, iy, varid;
01672
01673
        /\star Check if variable exists... \star/
01674
        if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
01675
         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
01676
            return:
01677
01678
       /* Read data... */
```

```
NC(nc_get_var_float(ncid, varid, help));
01680
01681
           /* Copy and check data... */
01682
           for (ix = 0; ix < met->nx; ix++)
            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>
01683
01684
01685
01686
01687
                     dest[ix][iy][ip] *= scl;
01688
                  else
01689
                      dest[ix][iy][ip] = GSL_NAN;
                }
01690
01691 }
```

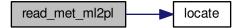
5.19.2.27 void read\_met\_ml2pl (  $ctl_t * ctl$ ,  $met_t * met$ , float var[EX][EY][EP] )

Convert meteorological data from model levels to pressure levels.

Definition at line 1695 of file libtrac.c.

```
01698
01699
01700
        double aux[EP], p[EP], pt;
01701
01702
        int ip, ip2, ix, iy;
01704
         /* Loop over columns... */
01705
        for (ix = 0; ix < met->nx; ix++)
01706
          for (iy = 0; iy < met->ny; iy++) {
01707
01708
             /* Copy pressure profile... */
for (ip = 0; ip < met->np; ip++)
01709
01710
               p[ip] = met->pl[ix][iy][ip];
01711
             /* 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]))
01712
01713
01714
01716
                 pt = p[0];
01717
               else if ((pt > p[met->np - 1] && p[1] > p[0])
01718
                          || (pt < p[met->np - 1] && p[1] < p[0]))
              01719
01720
01721
01722
01723
01724
             /* Copy data... */
for (ip = 0; ip < ctl->met_np; ip++)
01725
01726
01727
               var[ix][iy][ip] = (float) aux[ip];
01728
01729 }
```

Here is the call graph for this function:



```
5.19.2.28 void read_met_periodic ( met_t * met )
```

Create meteorological data with periodic boundary conditions.

Definition at line 1733 of file libtrac.c.

```
01734
01735
01736
           int ip, iv;
01738
            /* Check longitudes... */
01739
            if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
                             + met - > lon[1] - met - > lon[0] - 360) < 0.01))
01740
01741
01742
01743
            /* Increase longitude counter... */
            if ((++met->nx) > EX)
01744
01745
              ERRMSG("Cannot create periodic boundary conditions!");
01746
           /* Set longitude... */
met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
01747
01748
        lon[0];
01749
01750
             /* Loop over latitudes and pressure levels... */
            for (iy = 0; iy < met->ny; iy++)
  for (ip = 0; ip < met->np; ip++) {
01751
01752
                 met->ps[met->nx - 1][iy] = met->ps[0][iy];
met->pt[met->nx - 1][iy] = met->pt[0][iy];
met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
01753
01754
01755
                 met->z[met->nx - 1][iy][ip] = met->z[i0][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];

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

met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
01756
01757
01758
01759
                 met->pv[met->nx - 1][iy][ip] = met->pv[0][iy][ip];
met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
01760
01761
01762
                  met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01763
01764 }
```

## 5.19.2.29 void read\_met\_pv ( met\_t \* met )

Calculate potential vorticity.

Definition at line 1768 of file libtrac.c.

```
01769
01770
01771
        double c0, c1, cr, dx, dy, dp, dtdx, dvdx, dtdy, dudy, dtdp, dudp, dvdp,
01772
          latr, vort, pows[EP];
01774
        int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
01775
         /* Set powers... */
for (ip = 0; ip < met->np; ip++)
01776
01777
          pows[ip] = pow(1000. / met->p[ip], 0.286);
01778
01779
01780
         /* Loop over grid points... */
01781
         for (ix = 0; ix < met->nx; ix++) {
01782
          /* Set indices... */
ix0 = GSL_MAX(ix - 1, 0);
01783
01784
01785
          ix1 = GSL_MIN(ix + 1, met -> nx - 1);
01786
01787
           /* Loop over grid points... */
01788
           for (iy = 0; iy < met->ny; iy++) {
01789
             /* Set indices... */
iy0 = GSL_MAX(iy - 1, 0);
01790
01791
             iy1 = GSL_MIN(iy + 1, met -> ny - 1);
01792
01793
01794
              /* Set auxiliary variables... */
             latr = GSL_MIN(GSL_MAX(met->lat[iy], -89.), 89.);
01795
01796
             dx = 1000. * deg2dx(met->lon[ix1] - met->lon[ix0], latr);
dy = 1000. * deg2dy(met->lat[iy1] - met->lat[iy0]);
01797
01798
             c0 = cos(met->lat[iy0] / 180. * M_PI);
01799
             c1 = cos(met->lat[iy1] / 180. * M_PI);
```

```
cr = cos(latr / 180. * M_PI);
             vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
01802
01803
              /\star Loop over grid points... \star/
             for (ip = 0; ip < met->np; ip++) {
01804
01805
01806
                /* Set indices... */
01807
                ip0 = GSL\_MAX(ip - 1, 0);
               ip1 = GSL_MIN(ip + 1, met->np - 1);
01808
01809
01810
                /* Set auxiliary variables... */
01811
                dp = 100. * (met->p[ip1] - met->p[ip0]);
01812
01813
                /* 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;
01814
01815
01816
                /* Get gradients in latitude... */
01817
                dtdy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
01818
                dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
01819
01820
01821
                /\star Get gradients in pressure... \star/
01822
                dtdp =
                (met->t[ix][iy][ip1] * pows[ip1] -
  met->t[ix][iy][ip0] * pows[ip0]) / dp;
dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / dp;
01823
01824
01825
01826
                dvdp = (met -> v[ix][iy][ip1] - met -> v[ix][iy][ip0]) / dp;
01827
01828
                /* Calculate PV... */
01829
               met->pv[ix][iy][ip] = (float)
  (1e6 * G0 *
01830
01831
                    (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
01832
01833
01834 }
01835 }
```

Here is the call graph for this function:

```
5.19.2.30 void read_met_sample ( ctl_t * ctl, met_t * met )
```

Downsampling of meteorological data.

Definition at line 1839 of file libtrac.c.

```
01841
                          {
01842
01843
         met_t *help;
01844
01845
         float w, wsum;
01846
01847
         int ip, ip2, ix, ix2, iy, iy2;
01848
01849
         /* Check parameters... */
01850
         if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1)</pre>
01851
           return;
01852
         /* Allocate... */
01853
01854
         ALLOC(help, met t, 1);
01856
          /* Copy data... */
01857
         help->nx = met->nx;
         help->ny = met->ny;
01858
         help->np = met->np;
01859
         memcpy(help->lon, met->lon, sizeof(met->lon));
memcpy(help->lat, met->lat, sizeof(met->lat));
01860
01861
01862
         memcpy(help->p, met->p, sizeof(met->p));
01863
01864
         /* Smoothing... */
         for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
01865
           for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
    for (ip = 0; ip < met->np; ip += ctl->met_dp) {
01866
01867
01868
                 help \rightarrow ps[ix][iy] = 0;
                 help \rightarrow pt[ix][iy] = 0;
01869
01870
                 help \rightarrow z[ix][iy][ip] = 0;
01871
                 help \rightarrow t[ix][iy][ip] = 0;
                help->u[ix][iy][ip] = 0;
help->v[ix][iy][ip] = 0;
01872
01873
01874
                help \rightarrow w[ix][iy][ip] = 0;
```

```
help \rightarrow pv[ix][iy][ip] = 0;
01876
                 help->h2o[ix][iy][ip] = 0;
01877
                 help \rightarrow 03[ix][iy][ip] = 0;
01878
                 wsum = 0;
                 for (ix2 = GSL_MAX(ix - ctl->met_sx + 1, 0);
    ix2 <= GSL_MIN(ix + ctl->met_sx - 1, met->nx - 1); ix2++)
    for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
01879
01880
01881
                         iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
01882
                      for (ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
01883
                        ip2 - GSI_MAX(ip - Ctl->met_sp + 1, 0);
ip2 <= GSI_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)
  * (float) (1.0 - fabs(ip - ip2) / ctl->met_sp);
01884
01885
01886
01887
                        help \rightarrow ps[ix][iy] += w * met \rightarrow ps[ix2][iy2];
01888
                         help->pt[ix][iy] += w * met->pt[ix2][iy2];
01889
                        help->z[ix][iy][ip] += w * met->z[ix2][iy2][ip2];
help->t[ix][iy][ip] += w * met->t[ix2][iy2][ip2];
01890
01891
                        help->u[ix][iy][ip] += w * met->u[ix2][iy2][ip2];
01892
                        help->v[ix][iy][ip] += w * met->v[ix2][iy2][ip2];
01893
                        help \rightarrow w[ix][iy][ip] += w * met \rightarrow w[ix2][iy2][ip2];
01894
01895
                        help \rightarrow pv[ix][iy][ip] += w * met \rightarrow pv[ix2][iy2][ip2];
01896
                        help->h2o[ix][iy][ip] += w * met->h2o[ix2][iy2][ip2];
                        help->o3[ix][iy][ip] += w * met->o3[ix2][iy2][ip2];
01897
01898
                        wsum += w:
01899
01900
                 help->ps[ix][iy] /= wsum;
01901
                 help->pt[ix][iy] /= wsum;
01902
                 help->t[ix][iy][ip] /= wsum;
                 help->z[ix][iy][ip] /= wsum;
01903
                 help->u[ix][iy][ip] /= wsum;
01904
                 help->v[ix][iy][ip] /= wsum;
01905
01906
                 help->w[ix][iy][ip] /= wsum;
01907
                 help->pv[ix][iy][ip] /= wsum;
01908
                 help->h2o[ix][iy][ip] /= wsum;
01909
                 help->03[ix][iy][ip] /= wsum;
01910
01911
           }
01912
01913
01914
          /* Downsampling... */
         met->nx = 0;
for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
01915
01916
           met->lon[met->nx] = help->lon[ix];
01917
            met->ny = 0;
for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
01918
01919
01920
              met->lat[met->ny] = help->lat[iy];
              met->ps[met->nx][met->ny] = help->ps[ix][iy];
met->pt[met->nx][met->ny] = help->pt[ix][iy];
01921
01922
               met \rightarrow np = 0;
01923
               for (ip = 0; ip < help->np; ip += ctl->met_dp) {
01924
01925
                 met->p[met->np] = help->p[ip];
01926
                 met->z[met->nx][met->ny][met->np] = help->z[ix][iy][ip];
01927
                 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];
01928
                 met->v[met->nx] [met->ny] [met->np] = help->v[ix][iy][ip];
01929
                 met->w[met->nx] [met->ny] [met->np] = help->w[ix][iy][ip];
01930
                 met->pv[met->nx][met->ny][met->np] = help->pv[ix][iy][ip];
01931
01932
                 met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
01933
                 met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
01934
                 met->np++;
01935
01936
              met->ny++;
01937
01938
            met->nx++;
01939
01940
01941
          /* Free... */
         free(help);
01942
01943 }
```

5.19.2.31 void read\_met\_tropo ( ctl\_t \* ctl, met\_t \* met )

Calculate tropopause pressure.

Definition at line 1947 of file libtrac.c.

```
01949 {
01950
01951 gsl_interp_accel *acc;
01952
```

```
gsl_spline *spline;
01954
01955
        double p2[400], pv[400], pv2[400], t[400], t2[400], th[400], th2[400],
01956
         z[400], z2[400];
01957
01958
        int found, ix, iv, iz, iz2;
01959
01960
        /* Allocate... */
01961
        acc = gsl_interp_accel_alloc();
01962
        spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) met->np);
01963
01964
        /* Get altitude and pressure profiles... */
        for (iz = 0; iz < met->np; iz++)
01965
01966
          z[iz] = Z(met->p[iz]);
01967
         for (iz = 0; iz <= 170; iz++)
         z2[iz] = 4.5 + 0.1 * iz;
01968
          p2[iz] = P(z2[iz]);
01969
01970
01971
01972
         /* Do not calculate tropopause... */
01973
         if (ctl->met_tropo == 0)
          for (ix = 0; ix < met->nx; ix++)
01974
             for (iy = 0; iy < met->ny; iy++)
01975
01976
               met->pt[ix][iy] = GSL_NAN;
01977
01978
        /* Use tropopause climatology... */
        else if (ctl->met_tropo == 1)
01979
01980
         for (ix = 0; ix < met->nx; ix++)
01981
             for (iy = 0; iy < met->ny; iy++)
01982
               met->pt[ix][iy] = clim_tropo(met->time, met->lat[iy]);
01983
01984
        /* Use cold point... */
01985
        else if (ctl->met_tropo == 2) {
01986
          /* Loop over grid points... */
for (ix = 0; ix < met->nx; ix++)
01987
01988
             for (iy = 0; iy < met->ny; iy++) {
01989
01990
01991
                /* Interpolate temperature profile... */
               for (iz = 0; iz < met->np; iz++)
   t[iz] = met->t[ix][iy][iz];
01992
01993
               gsl_spline_init(spline, z, t, (size_t) met->np);
for (iz = 0; iz <= 170; iz++)</pre>
01994
01995
01996
                 t2[iz] = gsl_spline_eval(spline, z2[iz], acc);
01997
01998
                /* Find minimum...
               iz = (int) gsl_stats_min_index(t2, 1, 171);
if (iz <= 0 || iz >= 170)
01999
02000
                 met->pt[ix][iy] = GSL_NAN;
02001
02002
               else
02003
                 met \rightarrow pt[ix][iy] = p2[iz];
02004
02005
02006
02007
        /* Use WMO definition... */
02008
        else if (ctl->met tropo == 3 || ctl->met tropo == 4) {
02009
02010
           /* Loop over grid points... */
02011
          for (ix = 0; ix < met->nx; ix++)
02012
             for (iy = 0; iy < met->ny; iy++) {
02013
02014
               /\star Interpolate temperature profile... \star/
               for (iz = 0; iz < met->np; iz++)
  t[iz] = met->t[ix][iy][iz];
02015
02016
02017
               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>
02018
02019
02020
               /* Find 1st tropopause... */
met->pt[ix][iy] = GSL_NAN;
02021
               for (iz = 0; iz <= 140; iz++) {
  found = 1;
02023
02024
                  for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
if (1000. * G0 / RA * log(t2[iz2] / t2[iz])</pre>
02025
02026
                         / log(p2[iz2] / p2[iz]) > 2.0) {
02027
02028
                      found = 0;
02029
                      break;
02030
02031
                  if (found) {
                   if (iz > 0 && iz < 140)
02032
                     met->pt[ix][iy] = p2[iz];
02033
02034
                    break;
02035
02036
02037
               /* Find 2nd tropopause... */
02038
02039
               if (ctl->met_tropo == 4) {
```

```
met->pt[ix][iy] = GSL_NAN;
02041
                 for (; iz <= 140; iz++) {
                   found = 1;
02042
                   02043
02044
02045
                        found = 0;
02047
                        break;
02048
02049
                   if (found)
02050
                     break;
02051
02052
                 for (; iz <= 140; iz++) {</pre>
02053
                   found = 1;
02054
                    for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)</pre>
                     if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
     / log(p2[iz2] / p2[iz]) > 2.0) {
02055
02056
02057
                        found = 0;
02058
                        break;
02060
                    if (found) {
02061
                      if (iz > 0 && iz < 140)
                       met \rightarrow pt[ix][iy] = p2[iz];
02062
02063
                      break:
02064
                   }
02065
                 }
02066
               }
             }
02067
02068
        }
02069
02070
        /* Use dynamical tropopause... */
        else if (ctl->met_tropo == 5) {
02072
02073
           /\star Loop over grid points... \star/
02074
          for (ix = 0; ix < met->nx; ix++)
             for (iy = 0; iy < met->ny; iy++) {
02075
02076
               /* Interpolate potential vorticity profile... */
02078
               for (iz = 0; iz < met->np; iz++)
02079
                 pv[iz] = met->pv[ix][iy][iz];
02080
               gsl_spline_init(spline, z, pv, (size_t) met->np);
02081
               for (iz = 0; iz \leq 160; iz++)
02082
                 pv2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02083
02084
               /* Interpolate potential temperature profile... */
02085
               for (iz = 0; iz < met->np; iz++)
02086
                 th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
               gsl_spline_init(spline, z, th, (size_t) met->np);
for (iz = 0; iz <= 160; iz++)
  th2[iz] = gsl_spline_eval(spline, z2[iz], acc);</pre>
02087
02088
02089
02090
02091
                /\star Find dynamical tropopause 3.5 PVU + 380 K \star/
02092
               met->pt[ix][iy] = GSL_NAN;
               for (iz = 0; iz <= 160; iz++)

if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
02093
02094
02095
                  if (iz > 0 && iz < 160)
                     met->pt[ix][iy] = p2[iz];
02097
                   break;
02098
02099
             }
02100
        }
02101
02102
        else
02103
          ERRMSG("Cannot calculate tropopause!");
02104
        /* Free... */
02105
02106
        gsl_spline_free(spline);
02107
        gsl_interp_accel_free(acc);
02108 }
```

Here is the call graph for this function:

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

Read a control parameter from file or command line.

Definition at line 2112 of file libtrac.c.

```
02119
02120
02121
        FILE *in = NULL;
02122
02123
         char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
02124
          msg[2 * LEN], rvarname[LEN], rval[LEN];
02125
02126
02127
02128
         /* Open file... */
        if (filename[strlen(filename) - 1] != '-')
  if (!(in = fopen(filename, "r")))
02129
02130
             ERRMSG("Cannot open file!");
02131
02132
02133
         /* Set full variable name... */
02134
        if (arridx >= 0) {
         sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
02135
02136
02137
        } else {
          sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
02138
02139
02140
02141
02142
        /* Read data... */
02143
        if (in != NULL)
         while (fgets(line, LEN, in))
02144
02145
            if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
              if (strcasecmp(rvarname, fullname1) == 0 ||
02146
02147
                   strcasecmp(rvarname, fullname2) == 0) {
02148
                 contain = 1;
02149
                 break:
02150
               }
02151
        for (i = 1; i < argc - 1; i++)</pre>
         if (strcasecmp(argv[i], fullname1) == 0 ||
02152
             strcasecmp(argv[i], fullname2) == 0) {
sprintf(rval, "%s", argv[i + 1]);
02153
02154
02155
             contain = 1;
02156
             break;
02157
          }
02158
02159
        /* Close file... */
        if (in != NULL)
02160
02161
         fclose(in);
02162
        /* Check for missing variables... */
02163
02164
        if (!contain) {
         if (strlen(defvalue) > 0)
   sprintf(rval, "%s", defvalue);
02165
02166
           else {
02167
02168
            sprintf(msg, "Missing variable %s!\n", fullname1);
02169
             ERRMSG (msg);
02170
02171
02172
02173
        /* Write info... */
02174 printf("%s = %s\n", fullname1, rval);
02176
        /* Return values... */
02177
        if (value != NULL)
          sprintf(value, "%s", rval);
02178
02179
        return atof(rval);
02180 }
```

5.19.2.33 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 2184 of file libtrac.c.

```
02192
02193
        struct tm t0, t1;
02194
02195
02196
        t0.tm_year = 100;
02197
        t0.tm_mon = 0;
        t0.tm_mday = 1;
t0.tm_hour = 0;
02198
02199
        t0.tm_min = 0;
02200
02201
        t0.tm\_sec = 0;
02202
```

```
02203    t1.tm_year = year - 1900;
02204    t1.tm_mon = mon - 1;
02205    t1.tm_mday = day;
02206    t1.tm_bour = hour;
02207    t1.tm_min = min;
02208    t1.tm_sec = sec;
02209
02210    *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
02211 }
```

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

Measure wall-clock time.

Definition at line 2215 of file libtrac.c.

```
02218
                   {
02219
02220
        static double starttime[NTIMER], runtime[NTIMER];
02221
02222
        /* Check id... */
        if (id < 0 || id >= NTIMER)
02224
          ERRMSG("Too many timers!");
02225
02226
        /* Start timer... */
02227
        if (mode == 1) {
        if (starttime[id] <= 0)</pre>
02228
            starttime[id] = omp_get_wtime();
02230
02231
            ERRMSG("Timer already started!");
02232
02233
        /* Stop timer... */
else if (mode == 2) {
02234
02235
        if (starttime[id] > 0) {
02236
02237
            runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
02238
            starttime[id] = -1;
02239
02240 }
02241
02242
        /* Print timer... */
        else if (mode == 3) {
    printf("%s = %.3f s\n", name, runtime[id]);
02243
02244
          runtime[id] = 0;
02245
        }
02246
02247 }
```

5.19.2.35 void write\_atm ( const char \* filename, ctl\_t \* ctl, atm\_t \* atm, double t )

Write atmospheric data.

Definition at line 2251 of file libtrac.c.

```
02255
                   {
02257
        FILE *in, *out;
02258
02259
        char line[LEN];
02260
02261
        double r, t0, t1;
02262
02263
        int ip, iq, year, mon, day, hour, min, sec;
02264
02265
        /\star Set time interval for output... \star/
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
02266
02267
02268
02269
        /* Write info... */
02270
        printf("Write atmospheric data: %s\n", filename);
02271
        /* Write ASCII data... */
02272
02273
        if (ctl->atm_type == 0) {
02274
          /* Check if gnuplot output is requested... */
```

```
02276
          if (ctl->atm_gpfile[0] != '-') {
02277
            /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
02278
02279
              ERRMSG("Cannot create pipe to gnuplot!");
02280
02281
            /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
02283
02284
02285
            /* Set time string... */
            jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02286
02287
02288
                     year, mon, day, hour, min);
02289
02290
            /\star Dump gnuplot file to pipe... \star/
            if (!(in = fopen(ctl->atm_gpfile, "r")))
    ERRMSG("Cannot open file!");
02291
02292
            while (fgets(line, LEN, in))
fprintf(out, "%s", line);
02293
02295
            fclose(in);
02296
02297
02298
          else {
02299
02300
            /* Create file... */
02301
            if (!(out = fopen(filename, "w")))
02302
              ERRMSG("Cannot create file!");
02303
02304
          /* Write header... */
02305
02306
          fprintf(out,
02307
                   "# $1 = time [s]\n"
02308
                  "# $2 = altitude [km] \n"
                  "# $3 = longitude [deg] \n" "# $4 = latitude [deg] \n");
02309
          02310
02311
02312
02313
02314
02315
          /* Write data... */
02316
          for (ip = 0; ip < atm->np; ip++) {
02317
02318
            /* Check time... */
            if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
02319
02320
              continue;
02321
            02322
02323
02324
02325
02326
02327
              fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02328
02329
            fprintf(out, "\n");
02330
02331
02332
          /* Close file... */
02333
          fclose(out);
02334
02335
02336
       /* Write binary data... */
02337
       else if (ctl->atm_type == 1) {
02338
02339
          /* Create file... */
02340
          if (!(out = fopen(filename, "w")))
            ERRMSG("Cannot create file!");
02341
02342
          /* Write data... */
02343
02344
          FWRITE(&atm->np, int,
                 1,
02346
                 out);
02347
          FWRITE(atm->time, double,
02348
                   (size_t) atm->np,
02349
                 out);
02350
          FWRITE(atm->p, double,
02351
                  (size_t) atm->np,
02352
                 out);
02353
          FWRITE(atm->lon, double,
02354
                   (size_t) atm->np,
                 out);
02355
          FWRITE(atm->lat, double,
02356
02357
                   (size_t) atm->np,
02358
                 out);
02359
          for (iq = 0; iq < ctl->nq; iq++)
02360
           FWRITE(atm->q[iq], double,
02361
                     (size_t) atm->np,
02362
                   out);
```

Here is the call graph for this function:



5.19.2.36 void write\_csi ( const char \* filename, ctl\_t \* ctl, atm\_t \* atm, double t )

Write CSI data.

Definition at line 2375 of file libtrac.c.

```
02380
02381
         static FILE *in, *out;
02382
02383
         static char line[LEN];
02384
02385
         static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
02386
           rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
02387
02388
         static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
02389
02390
         /* Init... */
02391
         if (t == ctl->t_start) {
02392
02393
            /\star Check quantity index for mass... \star/
02394
           if (ctl->qnt_m < 0)
   ERRMSG("Need quantity mass!");</pre>
02395
02396
02397
            /* Open observation data file... */
           printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
if (!(in = fopen(ctl->csi_obsfile, "r")))
    ERRMSG("Cannot open file!");
02398
02399
02400
02401
02402
            /* Create new file... */
           printf("Write CSI data: %s\n", filename);
if (!(out = fopen(filename, "w")))
02403
02404
02405
              ERRMSG("Cannot create file!");
02406
            /* Write header... */
02407
02408
            fprintf(out,
02409
                      "# $1 = time [s] \n"
02410
                      "# $2 = number of hits (cx)\n"
02411
                      "# $3 = number of misses (cy) \n"
                      "# $4 = number of false alarms (cz)\n"
02412
                     "# $5 = number of observations (cx + cy)\n"
"# $6 = number of forecasts (cx + cz)\n"
02413
02414
                     "# $7 = bias (forecasts/observations) [%%]\n"
02415
                     "# $8 = probability of detection (POD) [%%]\n" # $9 = false alarm rate (FAR) [%%]\n"
02416
02417
                     "# $10 = critical success index (CSI) [%%]\n\n");
02418
02419
02420
02421
         /* Set time interval... */
02422
         t0 = t - 0.5 * ct1->dt_mod;
```

```
t1 = t + 0.5 * ctl->dt_mod;
02424
02425
         /* Initialize grid cells... */
02426
         for (ix = 0; ix < ctl->csi_nx; ix++)
02427
          for (iy = 0; iy < ctl->csi_ny; iy++)
  for (iz = 0; iz < ctl->csi_nz; iz++)
02428
               modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
02430
02431
         /* Read observation data... */
02432
         while (fgets(line, LEN, in)) {
02433
           /* Read data... */
if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02434
02435
02436
02437
              continue;
02438
           /\star Check time... \star/
02439
           if (rt < t0)</pre>
02440
02441
             continue;
           if (rt > t1)
02442
02443
            break;
02444
02445
           /* Calculate indices... */
           ix = (int) ((rlon - ctl->csi\_lon0))
02446
02447
                         / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
           iy = (int) ((rlat - ctl->csi_lat0)
02449
                         / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02450
           iz = (int) ((rz - ctl->csi_z0)
02451
                         / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02452
02453
           /* Check indices... */
02454
           if (ix < 0 || ix >= ctl->csi_nx ||
02455
                iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02456
02457
02458
           /* Get mean observation index... */
02459
           obsmean[ix][iy][iz] += robs;
02460
           obscount[ix][iy][iz]++;
02461
02462
02463
         /* Analyze model data... ∗/
02464
         for (ip = 0; ip < atm->np; ip++) {
02465
02466
           /* Check time... */
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
02467
02468
02469
02470
           /* Get indices... */
02471
           ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
                         / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02472
           iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
02474
                          (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
           02475
02476
02477
02478
           /* Check indices... */
           if (ix < 0 || ix >= ctl->csi_nx ||
02480
               iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02481
02482
           /\star Get total mass in grid cell... \star/
02483
           modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02484
02485
02486
02487
         /* Analyze all grid cells... */
02488
         for (ix = 0; ix < ctl->csi_nx; ix++)
02489
          for (iy = 0; iy < ctl->csi_ny; iy++)
  for (iz = 0; iz < ctl->csi_nz; iz++) {
02490
02491
02492
                /* Calculate mean observation index... */
02493
                if (obscount[ix][iy][iz] > 0)
02494
                  obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
02495
                /\star Calculate column density... \star/
02496
                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;
02497
02498
02499
                 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);
02500
02501
02502
02503
02504
02505
02506
                /* Calculate CSI... */
02507
                if (obscount[ix][iy][iz] > 0) {
                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
    modmean[ix][iy][iz] >= ctl->csi_modmin)
02508
02509
```

```
cx++;
02511
                  else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02512
                              modmean[ix][iy][iz] < ctl->csi_modmin)
02513
                    cv++;
02514
                  else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
    modmean[ix][iy][iz] >= ctl->csi_modmin)
02515
02516
                     cz++;
02517
02518
             }
02519
         /* Write output... */
if (fmod(t, ctl->csi_dt_out) == 0) {
02520
02521
02522
           02523
02524
                     (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,
02525
02526
02527
02528
                     (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
02530
02531
            /\star Set counters to zero... \star/
02532
           cx = cy = cz = 0;
02533
02534
        /* Close file... */
02535
         if (t == ctl->t_stop)
02536
02537
           fclose(out);
02538 }
```

5.19.2.37 void write\_ens ( const char \* filename, ctl\_t \* ctl, atm\_t \* atm, double t )

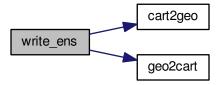
Write ensemble data.

Definition at line 2542 of file libtrac.c.

```
02547
02548
        static FILE *out;
02549
        static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
02550
02551
         t0, t1, x[NENS][3], xm[3];
02552
02553
        static int ip, iq;
02554
02555
        static size_t i, n;
02556
02557
        /* Init... */
02558
        if (t == ctl->t_start) {
02559
02560
          /* Check quantities... */
02561
          if (ctl->qnt_ens < 0)
   ERRMSG("Missing ensemble IDs!");</pre>
02562
02563
02564
          /\star Create new file... \star/
          printf("Write ensemble data: %s\n", filename);
if (!(out = fopen(filename, "w")))
02565
02566
            ERRMSG("Cannot create file!");
02567
02568
02569
          /* Write header... */
02570
          fprintf(out,
02571
                   "# $1 = time [s] \n"
02572
                   "# $2 = altitude [km] \n"
                   "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02573
          for (iq = 0; iq < ctl->nq; iq+)
fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
02574
02575
                     ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02576
          02577
02578
02579
02580
02581
02582
02583
        /* Set time interval... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
02584
02585
02586
02587
        /* Init...
02588
        ens = GSL_NAN;
02589
        n = 0;
```

```
02590
02591
         /* Loop over air parcels... */
02592
         for (ip = 0; ip < atm->np; ip++) {
02593
02594
           /* Check time... */
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
02595
02596
             continue;
02597
02598
           /* Check ensemble id... */
02599
           if (atm->q[ctl->qnt_ens][ip] != ens) {
02600
02601
             /* Write results... */
02602
             if (n > 0) {
02603
02604
                /∗ Get mean position...
                for (i = 0; i < n; i++) {
    xm[0] += x[i][0] / (double) n;
    xm[1] += x[i][1] / (double) n;</pre>
02605
02606
02607
02608
                  xm[2] += x[i][2] / (double) n;
02609
02610
02611
                cart2geo(xm, &dummy, &lon, &lat);
                02612
02613
                         lat):
02614
02615
                /* Get quantity statistics... */
                for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02616
02617
02618
                  fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02619
                for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02620
02621
02622
                  fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02623
02624
                fprintf(out, " lu\n", n);
02625
02626
02627
             /* Init new ensemble... */
02628
             ens = atm->q[ctl->qnt_ens][ip];
02629
             n = 0;
02630
02631
           /* Save data... */
02632
02633
           p[n] = atm -> p[ip];
           geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
02634
02635
           for (iq = 0; iq < ctl->nq; iq++)
           q[iq][n] = atm->q[iq][ip];
if ((++n) >= NENS)
02636
02637
             ERRMSG("Too many data points!");
02638
02639
02640
02641
         /* Write results... */
02642
         if (n > 0) {
02643
02644
           /* Get mean position... */
           xm[0] = xm[1] = xm[2] = 0;
for (i = 0; i < n; i++) {
02645
02646
             xm[0] += x[i][0] / (double) n;
xm[1] += x[i][1] / (double) n;
xm[2] += x[i][2] / (double) n;
02647
02648
02649
02650
           cart2geo(xm, &dummy, &lon, &lat);
fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
02651
02652
02653
02654
           /* Get quantity statistics... */
           for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02655
02656
02657
             fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02658
           for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02659
02660
02661
              fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02662
           fprintf(out, " lu\n", n);
02663
02664
02665
02666
         /* Close file... */
02667
         if (t == ctl->t_stop)
02668
           fclose(out);
02669 }
```

Here is the call graph for this function:



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

Write gridded data.

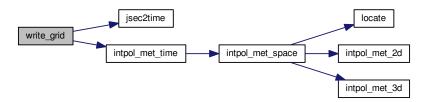
Definition at line 2673 of file libtrac.c.

```
02679
                       {
02681
         FILE *in, *out;
02682
02683
         char line[LEN];
02684
02685
         static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
02686
            area, rho_air, press, temp, cd, vmr, t0, t1, r;
02687
02688
         static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
02689
02690
          /* Check dimensions... */
          if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
02691
02692
            ERRMSG("Grid dimensions too large!");
02693
02694
          /\star Check quantity index for mass... \star/
02695
          if (ctl->qnt_m < 0)
02696
            ERRMSG("Need quantity mass!");
02697
02698
          /* Set time interval for output... */
02699
          t0 = t - 0.5 * ctl->dt_mod;
          t1 = t + 0.5 * ctl->dt_mod;
02700
02701
         /* Set grid box size... */
dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
02702
02703
02704
02705
          dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
02706
02707
          /* Initialize grid... */
         /* Initialize grid... */
for (ix = 0; ix < ctl->grid_nx; ix++)
    for (iy = 0; iy < ctl->grid_ny; iy++)
    for (iz = 0; iz < ctl->grid_nz; iz++)
02708
02709
02710
02711
                 mass[ix][iy][iz] = 0;
02712
02713
          /* Average data... */
          for (ip = 0; ip < atm->np; ip++)
  if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {</pre>
02714
02715
02716
02717
               /* 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);
02718
02719
02720
02721
               /* Check indices... */
if (ix < 0 || ix >= ctl->grid_nx ||
02722
02723
02724
                    iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
02725
02726
02727
               /* Add mass... */
mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02728
02729
02730
```

```
/* Check if gnuplot output is requested... */
02732
        if (ctl->grid_gpfile[0] != '-') {
02733
02734
           /\star Write info... \star/
02735
           printf("Plot grid data: %s.png\n", filename);
02736
02737
           /* Create gnuplot pipe... */
           if (!(out = popen("gnuplot", "w")))
02738
02739
             ERRMSG("Cannot create pipe to gnuplot!");
02740
           02741
02742
02743
02744
           /* Set time string... */
           jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02745
02746
02747
                    year, mon, day, hour, min);
02748
           /* Dump gnuplot file to pipe... */
02750
           if (!(in = fopen(ctl->grid_gpfile, "r")))
02751
             ERRMSG("Cannot open file!");
           while (fgets(line, LEN, in))
fprintf(out, "%s", line);
02752
02753
02754
           fclose(in);
02755
        }
02756
02757
02758
02759
           /* Write info... */
          printf("Write grid data: %s\n", filename);
02760
02761
02762
           /* Create file... */
02763
          if (!(out = fopen(filename, "w")))
02764
             ERRMSG("Cannot create file!");
02765
02766
         /* Write header... */
02767
02768
        fprintf(out,
02769
                  "# $1 = time [s] \n"
                  "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n"
"# $4 = latitude [deg]\n"
02770
02771
02772
                  "# $5 = surface area [km^2]\n"
"# $6 = layer width [km]\n"
02773
02774
02775
                  "# $7 = temperature [K] \n"
02776
                  "# $8 = column density [kg/m^2] n"
02777
                  "# $9 = volume mixing ratio [1]n\n");
02778
02779
         /* Write data... */
        for (ix = 0; ix < ctl->grid_nx; ix++) {
02780
         if
              (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
02782
             fprintf(out, "\n");
02783
               (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++)
02784
02785
02786
               if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
02788
                  /* Set coordinates... */
02789
                  z = ctl->grid_z0 + dz * (iz + 0.5);
lon = ctl->grid_lon0 + dlon * (ix + 0.5);
lat = ctl->grid_lat0 + dlat * (iy + 0.5);
02790
02791
02792
02794
                  /\star Get pressure and temperature... \star/
02795
                  press = P(z);
02796
                  intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
02797
                                    NULL, &temp, NULL, NULL, NULL, NULL, NULL, NULL);
02798
02799
                  /* Calculate surface area... */
                  * cos(lat * M_PI / 180.)
* cos(lat * M_PI / 180.)
02801
02802
                  /* Calculate column density... */
cd = mass[ix][iy][iz] / (le6 * area);
02803
02804
02805
02806
                  /* Calculate volume mixing ratio...
                  rho_air = 100. * press / (RA * temp);
vmr = MA / ctl->molmass * mass[ix][iy][iz]
02807
02808
02809
                    / (rho_air * 1e6 * area * 1e3 * dz);
02810
                  02811
02812
02813
02814
               }
02815
          }
02816
02817
```

```
02818  /* Close file... */
02819  fclose(out);
02820 }
```

Here is the call graph for this function:



5.19.2.39 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 2824 of file libtrac.c.

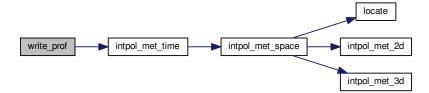
```
02830
                    {
02831
02832
         static FILE *in, *out;
02834
         static char line[LEN];
02835
         static double mass[GX][GY][GZ], obsmean[GX][GY], rt, rz, rlon, rlat, robs,
02836
02837
           t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, vmr, h2o,
02838
02839
02840
         static int obscount[GX][GY], ip, ix, iy, iz, okay;
02841
02842
         /* Init... */
         if (t == ctl->t_start) {
02843
02844
           /* Check quantity index for mass... */ if (ctl->qnt_m < 0)
02845
02846
02847
             ERRMSG("Need quantity mass!");
02848
           /* Check dimensions... */
if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
    ERRMSG("Grid dimensions too large!");
02849
02850
02851
02852
02853
           /\star Open observation data file... \star/
02854
           printf("Read profile observation data: %s\n", ctl->prof_obsfile);
02855
           if (!(in = fopen(ctl->prof_obsfile, "r")))
             ERRMSG("Cannot open file!");
02856
02857
02858
           /* Create new output file... */
           printf("Write profile data: %s\n", filename);
if (!(out = fopen(filename, "w")))
02859
02860
             ERRMSG("Cannot create file!");
02861
02862
02863
           /* Write header... */
02864
           fprintf(out,
02865
                            = time [s]\n"
02866
                    "# $2
                           = altitude [km] \n"
                    "# S3
                           = longitude [deg]\n"
02867
                    "# $4 = latitude [deg]\n"
02868
                    "# $5
                           = pressure [hPa]\n"
02869
02870
                    "# $6
                           = temperature [K]\n"
02871
                    "# $7
                           = volume mixing ratio [1]\n"
02872
                    "# $8 = H20 volume mixing ratio [1]\n"
                    "# $9 = 03 volume mixing ratio [1]\n"
"# $10 = mean BT index [K]\n");
02873
02874
02875
02876
           /* Set grid box size... */
02877
           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;
02879
02880
02881
02882
         /* Set time interval... */
t0 = t - 0.5 * ctl->dt_mod;
02883
         t1 = t + 0.5 * ctl->dt_mod;
02885
         /* Initialize... */
for (ix = 0; ix < ctl->prof_nx; ix++)
  for (iy = 0; iy < ctl->prof_ny; iy++) {
    obsmean[ix][iy] = 0;
02886
02887
02888
02889
02890
               obscount[ix][iy] = 0;
               for (iz = 0; iz < ctl->prof_nz; iz++)
02891
02892
                 mass[ix][iy][iz] = 0;
02893
02894
02895
         /* Read observation data... */
         while (fgets(line, LEN, in)) {
02896
02897
            /* Read data... */
if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02898
02899
02900
                 5)
02901
              continue;
02902
02903
            /* Check time... */
02904
            if (rt < t0)</pre>
02905
              continue;
            if (rt > t1)
02906
02907
              break:
02908
02909
            /* Calculate indices... */
           ix = (int) ((rlon - ctl->prof_lon0) / dlon);
iy = (int) ((rlat - ctl->prof_lat0) / dlat);
02910
02911
02912
           /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
02913
02914
              continue;
02916
02917
            /\star Get mean observation index... \star/
02918
           obsmean[ix][iy] += robs;
           obscount[ix][iy]++;
02919
02920
02921
02922
          /* Analyze model data... */
02923
          for (ip = 0; ip < atm->np; ip++) {
02924
           02925
02926
02927
              continue:
02928
02929
            /* Get indices... */
           ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
02930
02931
02932
02933
02934
            /* Check indices... */
02935
            if (ix < 0 || ix >= ctl->prof_nx ||
02936
                 iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
              continue;
02937
02938
           /* Get total mass in grid cell... */
mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02939
02940
02941
02942
          /* Extract profiles... */
02943
         for (ix = 0; ix < ctl->prof_nx; ix++)
  for (iy = 0; iy < ctl->prof_ny; iy++)
02944
02945
02946
              if (obscount[ix][iy] > 0) {
02948
                 /* Check profile... */
02949
                 okay = 0;
                 for (iz = 0; iz < ctl->prof_nz; iz++)
02950
                  if (mass[ix][iy][iz] > 0)
02951
02952
                     okay = 1;
02953
                 if (!okay)
02954
                   continue;
02955
                 /* Write output... */
fprintf(out, "\n");
02956
02957
02958
                 /* Loop over altitudes... */
02960
                 for (iz = 0; iz < ctl->prof_nz; iz++) {
02961
02962
                    /* Set coordinates... */
                   z = ctl->prof_z0 + dz * (iz + 0.5);
lon = ctl->prof_lon0 + dlon * (ix + 0.5);
02963
02964
```

```
lat = ctl - prof_lat0 + dlat * (iy + 0.5);
02966
02967
               /* Get pressure and temperature... */
02968
               press = P(z);
               02969
02970
02971
02972
               /* Calculate surface area... */
               area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
 * cos(lat * M_PI / 180.);
02973
02974
02975
02976
               /* Calculate volume mixing ratio... */
               rho_air = 100. * press / (RA * temp);

vmr = MA / ctl->molmass * mass[ix][iy][iz]
02977
02978
02979
                / (rho_air * area * dz * 1e9);
02980
               02981
02982
02983
02984
02985
02986
           }
02987
       /* Close file... */
if (t == ctl->t_stop)
02988
02989
         fclose(out);
02991 }
```

Here is the call graph for this function:



5.19.2.40 void write\_station ( const char \* filename, ctl\_t \* ctl, atm\_t \* atm, double t )

Write station data.

Definition at line 2995 of file libtrac.c.

```
02999
                    {
03000
03001
        static FILE *out;
03002
03003
        static double rmax2, t0, t1, x0[3], x1[3];
03004
03005
        static int ip, iq;
03006
03007
        /* Init... */
03008
        if (t == ctl->t_start) {
03009
03010
           /* Write info... */
03011
          printf("Write station data: %s\n", filename);
03012
          /* Create new file... */
if (!(out = fopen(filename, "w")))
03013
03014
             ERRMSG("Cannot create file!");
03015
03016
03017
           /* Write header... */
           fprintf(out,
03018
                    "# $1 = time [s] \n"
03019
                   "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03020
03021
03022
           for (iq = 0; iq < ctl->nq; iq++)
```

5.20 libtrac.c 97

```
fprintf(out, "# $\%i = \%s [\%s] \n", (iq + 5),
           ctl->qnt_name[iq], ctl->qnt_unit[iq]);
fprintf(out, "\n");
03024
03025
03026
03027
           /\star Set geolocation and search radius... \star/
03028
           geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
           rmax2 = gsl_pow_2(ctl->stat_r);
03029
03030
03031
        /* Set time interval for output... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
03032
03033
03034
03035
03036
         /* Loop over air parcels... */
03037
        for (ip = 0; ip < atm->np; ip++) {
03038
03039
           /\star Check time... \star/
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
03040
03041
             continue;
03042
03043
           /* Check station flag... */
03044
           if (ctl->qnt_stat >= 0)
           if (atm->q[ctl->qnt_stat][ip])
03045
03046
               continue;
03047
03048
           /* Get Cartesian coordinates... */
03049
           geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
03050
03051
           /\star Check horizontal distance... \star/
           if (DIST2(x0, x1) > rmax2)
03052
03053
            continue:
03054
03055
           /* Set station flag... */
03056
           if (ctl->qnt_stat >= 0)
03057
             atm->q[ctl->qnt_stat][ip] = 1;
03058
           /* Write data... */
fprintf(out, "%.2f %g %g %g",
03059
03060
03061
                   atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
           for (iq = 0; iq < ctl->nq; iq++) {
   fprintf(out, " ");
03062
03063
03064
             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
03065
03066
           fprintf(out, "\n");
03067
03068
03069
         /\star Close file... \star/
        if (t == ctl->t_stop)
03070
03071
           fclose(out);
03072 }
```

Here is the call graph for this function:



## 5.20 libtrac.c

```
00001 /*
00002
       This file is part of MPTRAC.
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
          Copyright (C) 2013-2018 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00028
00029 void cart2geo(
00030
         double *x,
          double *z,
00031
00032
         double *lon.
00033
         double *lat)
00034
00035
         double radius;
00036
         radius = NORM(x);
00037
         *lat = asin(x[2] / radius) * 180 / M_PI;
*lon = atan2(x[1], x[0]) * 180 / M_PI;
00038
00039
00040
          *z = radius - RE;
00041 }
00042
00044
00045 double clim hno3(
00046
          double t.
00047
          double lat,
00048
          double p) {
00049
00050
          static double secs[12] = { 1209600.00, 3888000.00, 6393600.00,
            9072000.00, 11664000.00, 14342400.00,
00051
             16934400.00, 19612800.00, 22291200.00,
00052
            24883200.00, 27561600.00, 30153600.00
00053
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
          static double hno3[12][18][10] = {
00064
            {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00065
              \{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
              \{0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181\},
00073
              {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},
00074
00075
00076
              {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
00080
              {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}, 
{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},
00084
00086
00087
              {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}, {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},
00088
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},
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
00100
              {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}, {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
00101
00102
```

5.20 libtrac.c 99

```
{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},
{0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
00105
00106
                {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
00107
                {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\},
00108
               {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},
00112
00113
               {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},
00114
00115
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}, {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
00118
00119
00120
00122
               {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},
00123
00124
                {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
00125
               {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
               {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
00130
                {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
00131
                {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02}
00132
                \{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},
00133
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
00136
                {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
              {\(1.13\), \(2.59\), \(7.49\), \(13.5\), \(15.4\), \(12.9\), \(11.3\), \(8.62\), \(4.18\), \(1.63\), \(\{0.973\), \(2.79\), \(7.23\), \(12.8\), \(15.2\), \(13.3\), \(11.6\), \(8.42\), \(4.06\), \(1.57\), \(\{1.46\), \(3.44\), \(6.78\), \(10.4\), \(12.7\), \(12.1\), \(10.5\), \(7.04\), \(3.59\), \(1.63\), \(\{1.52\), \(3.38\), \(6.04\), \(9.08\), \(11\), \(10.3\), \(8.9\), \(5.7\), \(2.77\), \(1.37\),
00137
00138
00139
00141
               {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}
00142
00143
                {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},
               {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},
00145
00146
               {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
00148
                {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473,
00149
                {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},
00150
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},
00154
00155
               {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},
00156
00157
               {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67}, {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
00158
               {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
00160
                {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217}
00161
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},
00164
00165
                {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302}
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}, {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
00168
00169
00170
               {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
00171
                {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},
00173
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}, {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},
00175
00176
00177
00179
00180
                {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
00181
                {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
00182
               {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
               {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25}
00183
                {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},
00185
00186
                {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
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}, {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
00188
00189
```

```
\{0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62\}\},
            00191
00192
              {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},
00195
              {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
              {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},
00199
              {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},
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
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
              {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}
00210
00211
              {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},
00212
00213
00214
              {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169), {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},
00218
              {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
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},
00220
              {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}, {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
00223
00224
              {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
00225
            {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
{{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
00226
00228
              {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
              {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
00229
              {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
              {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},
00232
              \{0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968\},
00234
00235
              {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
00236
              {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
00237
              {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}, {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
00238
00239
              \{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},
00241
00242
              {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52}
              {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}},
00243
00244
             {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
00245
              {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}, {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
00251
              {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},
00255
              {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},
00256
00257
              {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},
00260
00261
              {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},
{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},
00262
00263
00264
              {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
00266
00267
              {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45,
                                                                                    0.837
00268
              \{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}, {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},
00269
00270
              {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
00272
00273
              {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
00274
              {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}, {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
00275
00276
```

5.20 libtrac.c 101

```
{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
00281
00282
          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... */
00291
          ilat = locate(lats, 18, lat);
          ip = locate(ps, 10, p);
00292
00293
          isec = locate(secs, 12, sec);
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);
00301
          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 }
00308
00310
00311 double clim_tropo(
00312
          double t.
00313
          double lat) {
00315
          static double doys[12]
          = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00316
00317
00318
          static double lats[73]
            -8 -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, -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
00319
00320
00321
00322
00323
             15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
00324
             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
00325
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,
                    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,
00330
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
                    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
00339
            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, 98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
00340
00341
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
            287.5, 286.2, 285.81,
00344
           {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
00345
            297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
            161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3, 100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
00347
00348
           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},
00351
           {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
00353
           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,
00354
00355
00356
00357
            148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
            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},
00359
00360
           {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,
00361
00362
00363
```

```
101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
             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
             273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
00367
            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,
00368
00369
00371
00372
             105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
             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,
00376
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, 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,
00379
00380
00381
00383
             275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
00384
00385
            {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, 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,
00386
00387
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},
            {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, 243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
00393
00394
00395
             114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
00396
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}, {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
00400
00402
             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,
00403
00404
             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, 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,
00405
00406
00407
             279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
00408
             305.1},
00409
00410
            {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
             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,
00411
00412
             108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
00413
00414
             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,
00415
00416
             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}, {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 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,
00417
00418
00419
             100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
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,
00422
00423
00424
00425
             281.7, 281.1, 281.2}
00426
00427
00428
           double doy, p0, p1, pt;
00429
00430
           int imon, ilat;
00431
           /* Get day of year... */
doy = fmod(t / 86400., 365.25);
00432
00433
00434
           while (doy < 0)</pre>
00435
              doy += 365.25;
00436
            /* Get indices... */
00437
           imon = locate(doys, 12, doy);
ilat = locate(lats, 73, lat);
00438
00439
00440
00441
            /* Get tropopause pressure... */
           00442
00443
           00444
00445
           pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
00446
00447
00448
            /* Return tropopause pressure... */
00449
           return pt;
00450 }
```

5.20 libtrac.c 103

```
00453
00454 void day2doy(
00455
      int year,
00456
       int mon.
00457
      int day,
00458
      int *doy) {
00459
      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 \};
00460
00461
00462
00463
      /* Get day of year... */
if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
00464
00465
        *doy = d01[mon - 1] + day - 1;
00466
        *doy = d0[mon - 1] + day - 1;
00467
00468 }
00469
00471
00472 void doy2day(
00473
      int year,
00474
      int dov,
00475
       int *mon,
00476
      int *day) {
00477
      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 \};
00478
00479
00480
      int i:
00481
00482
       /* Get month and day... */
      if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
    for (i = 11; i >= 0; i--)
00483
00484
00485
         if (d01[i] <= doy)</pre>
        break;
*mon = i + 1;
00486
00487
        *day = doy - d01[i] + 1;
00488
00489
      } else {
00490
       for (i = 11; i >= 0; i--)
00491
        if (d0[i] <= doy)</pre>
        break;
*mon = i + 1;
00492
00493
00494
        *day = doy - d0[i] + 1;
00495
00496 }
00497
00499
00500 double deg2dx(
00501
      double dlon,
00502
      double lat) {
00503
00504
      return dlon * M_PI * RE / 180. * cos(lat / 180. * M_PI);
00505 }
00506
00508
00509 double deg2dy(
00510
      double dlat) {
00511
00512
       return dlat * M PI * RE / 180.;
00513 }
00514
00516
00517 double dp2dz(
00518
      double dp,
00519
      double p) {
00521
      return -dp * H0 / p;
00522 }
00523
00525
00526 double dx2deg(
      double dx,
00527
00528
      double lat) {
00529
      /* Avoid singularity at poles... */
if (lat < -89.999 || lat > 89.999)
00530
00531
00532
        return 0;
00533
00534
        return dx * 180. / (M_PI * RE * cos(lat / 180. * M_PI));
00535 }
00536
```

```
00538
00539 double dy2deg(
00540
       double dy)
00541
       return dy * 180. / (M_PI * RE);
00542
00543 }
00546
00547 double dz2dp(
00548
       double dz,
       double p) {
00549
00550
       return -dz * p / H0;
00551
00552 }
00553
00555
00556 void geo2cart(
00557
       double z,
00558
       double lon,
       double lat,
00559
00560
       double *x) {
00561
00562
       double radius;
00563
00564
       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);
00565
00566
00567
00568 }
00569
00571
00572 void get_met(
00573
      ctl_t * ctl,
char *metbase,
00574
00575
       double t,
00576
       met_t * met0,
00577
       met_t * met1)
00578
00579
       static int init;
00580
00581
       char filename[LEN];
00582
       /* Init... */
00583
00584
       if (t == ctl->t_start || !init) {
         init = 1;
00585
00586
         get_met_help(t, -1, metbase, ctl->dt_met, filename);
00587
         read_met(ctl, filename, met0);
00588
00589
00590
         get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
     dt_met, filename);
         read_met(ctl, filename, metl);
00591
00592
00593
00594
       /* Read new data for forward trajectories... */
00595
       if (t > met1->time && ct1->direction == 1) {
00596
         memcpy(met0, met1, sizeof(met_t));
         get_met_help(t, 1, metbase, ctl->dt_met, filename);
read_met(ctl, filename, met1);
00597
00598
00599
00600
00601
       /* Read new data for backward trajectories... */
00602
       if (t < met0->time && ctl->direction == -1) {
00603
        memcpy(met1, met0, sizeof(met_t));
get_met_help(t, -1, metbase, ctl->dt_met, filename);
00604
         read_met(ctl, filename, met0);
00605
00606
       }
00607 }
00608
00610
00611 void get_met_help(
00612
       double t,
00613
       int direct,
00614
       char *metbase,
00615
       double dt_met,
       char *filename) {
00616
00617
00618
       double t6, r;
00619
00620
       int year, mon, day, hour, min, sec;
00621
       /* Round time to fixed intervals... */
if (direct == -1)
00622
00623
```

5.20 libtrac.c 105

```
t6 = floor(t / dt_met) * dt_met;
00625
00626
        t6 = ceil(t / dt_met) * dt_met;
00627
00628
      /* Decode time... */
       jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00629
00630
00631
       sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", metbase, year, mon, day, hour);
00632
00633 }
00634
00636
00637 void intpol_met_2d(
00638
      double array[EX][EY],
00639
       int ix,
00640
       int iy,
00641
       double wx,
00642
       double wy,
00643
       double *var) {
00644
00645
       double aux00, aux01, aux10, aux11;
00646
00647
      /* Set variables...
00648
      aux00 = array[ix][iy];
      aux01 = array[ix][iy + 1];
00649
00650
       aux10 = array[ix + 1][iy];
00651
       aux11 = array[ix + 1][iy + 1];
00652
00653
      /* Interpolate horizontally... */
      aux00 = wy * (aux00 - aux01) + aux01;

aux11 = wy * (aux10 - aux11) + aux11;
00654
00655
00656
       *var = wx * (aux00 - aux11) + aux11;
00657 }
00658
00660
00661 void intpol_met_3d(
      float array[EX][EY][EP],
00663
       int ip,
00664
       int ix,
00665
       int iy,
00666
       double wp,
00667
       double wx,
00668
       double wy,
00669
       double *var) {
00670
00671
       double aux00, aux01, aux10, aux11;
00672
00673
      /* Interpolate vertically... */
       aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00674
00675
        + array[ix][iy][ip + 1];
00676
       aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
      00677
00678
00679
00680
00681
        + array[ix + 1][iy + 1][ip + 1];
00682
00683
      /* Interpolate horizontally... */
      aux00 = wy * (aux00 - aux01) + aux01;
aux11 = wy * (aux10 - aux11) + aux11;
*var = wx * (aux00 - aux11) + aux11;
00684
00685
00686
00687 }
00688
00690
00691 void intpol_met_space(
00692
      met t * met.
00693
       double p,
00694
       double lon,
00695
       double lat,
00696
       double *ps,
00697
       double *pt,
00698
       double *z,
00699
       double *t,
       double *u,
00700
00701
       double *v,
00702
       double *w.
00703
       double *pv,
00704
      double *h2o,
00705
      double *o3) {
00706
00707
      double wp, wx, wy;
00708
00709
       int ip, ix, iy;
00710
```

```
00711
       /* Check longitude... */
00712
       if (met->lon[met->nx - 1] > 180 && lon < 0)
00713
          lon += 360;
00714
00715
        /* Get indices... */
00716
       ip = locate(met->p, met->np, p);
       ix = locate(met->lon, met->nx, lon);
00718
        iy = locate(met->lat, met->ny, lat);
00719
       /* Get weights... */
wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
00720
00721
       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]);
00722
00723
00724
00725
        /* Interpolate...
00726
        if (ps != NULL)
00727
          intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
00728
        if (pt != NULL)
         intpol_met_2d(met->pt, ix, iy, wx, wy, pt);
00730
        if (z != NULL)
00731
          intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy, z);
00732
        if (t != NULL)
00733
         intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
00734
        if (u != NULL)
00735
          intpol_met_3d (met->u, ip, ix, iy, wp, wx, wy, u);
00736
        if (v != NULL)
00737
          intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
00738
       if (w != NULL)
00739
          intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
        if (pv != NULL)
00740
00741
         intpol_met_3d(met->pv, ip, ix, iy, wp, wx, wy, pv);
00742
        if (h2o != NULL)
00743
         intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
00744
        if (o3 != NULL)
00745
          intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00746 }
00747
00749
00750 void intpol_met_time(
00751
       met_t * met0,
met_t * met1,
00752
00753
       double ts.
00754
       double p,
00755
       double lon,
00756
       double lat,
00757
       double *ps,
00758
       double *pt,
00759
       double *z.
00760
       double *t.
00761
        double *u,
00762
       double *v,
00763
       double *w,
00764
       double *pv,
00765
       double *h2o.
00766
       double *o3) {
00767
00768
       double h2o0, h2o1, o30, o31, ps0, ps1, pt0, pt1, pv0, pv1, t0, t1, u0, u1,
00769
         v0, v1, w0, w1, wt, z0, z1;
00770
       /* Spatial interpolation... */
00771
       00772
00773
00774
                         pt == NULL ? NULL : &pt0,
00775
                          z == NULL ? NULL : &z0,
00776
                         t == NULL ? NULL : &t0,
00777
                         u == NULL ? NULL : &u0,
00778
                         v == NULL ? NULL : &v0.
00779
                         w == NULL ? NULL : &w0,
00780
                         pv == NULL ? NULL : &pv0,
00781
                         h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00782
        intpol_met_space(met1, p, lon, lat,
                         ps == NULL ? NULL : &ps1,
pt == NULL ? NULL : &pt1,
00783
00784
00785
                         z == NULL ? NULL : &z1,
                         t == NULL ? NULL : &t1,
00786
00787
                         u == NULL ? NULL : &u1,
00788
                         v == NULL ? NULL : &v1,
00789
                         w == NULL ? NULL : &w1,
00790
                         pv == NULL ? NULL : &pv1,
00791
                         h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00792
       /* Get weighting factor... */
wt = (met1->time - ts) / (met1->time - met0->time);
00793
00794
00795
00796
       /* Interpolate... */
00797
       if (ps != NULL)
```

```
*ps = wt * (ps0 - ps1) + ps1;
00799
       if (pt != NULL)
       *pt = wt * (pt0 - pt1) + pt1;
if (z != NULL)
00800
00801
00802
         *z = wt * (z0 - z1) + z1;
       if (t != NULL)
00803
         *t = wt * (t0 - t1) + t1;
00804
00805
       if (u != NULL)
       *u = wt * (u0 - u1) + u1;
if (v != NULL)
*v = wt * (v0 - v1) + v1;
00806
00807
00808
       if (w != NULL)
00809
       *w = wt * (w0 - w1) + w1;
if (pv != NULL)
00810
00811
00812
         *pv = wt * (pv0 - pv1) + pv1;
       if (h2o != NULL)
00813
         *h2o = wt * (h2o0 - h2o1) + h2o1;
00814
00815
       if (o3 != NULL)
         *03 = wt * (030 - 031) + 031;
00816
00817 }
00818
00820
00821 void jsec2time(
00822
       double jsec,
       int *year,
00824
       int *mon,
00825
       int *day,
00826
       int *hour,
00827
       int *min,
00828
       int *sec.
00829
       double *remain) {
00830
00831
       struct tm t0, *t1;
00832
00833
       time_t jsec0;
00834
00835
       t0.tm_year = 100;
00836
       t0.tm\_mon = 0;
       t0.tm_mday = 1;
t0.tm_hour = 0;
00837
00838
       t0.tm_min = 0;
t0.tm_sec = 0;
00839
00840
00841
00842
        jsec0 = (time_t) jsec + timegm(&t0);
00843
       t1 = gmtime(&jsec0);
00844
00845
       *year = t1->tm_year + 1900;
       *mon = t1->tm_mon + 1;
00846
       *day = t1->tm_mday;
00847
        *hour = t1->tm_hour;
00848
00849
        *min = t1->tm_min;
00850
       *sec = t1->tm_sec;
00851
       *remain = jsec - floor(jsec);
00852 }
00853
00855
00856 int locate(
00857
       double *xx,
00858
       int n,
00859
       double x) {
00860
00861
       int i, ilo, ihi;
00862
00863
       ilo = 0;
       ihi = n - 1;
00864
       i = (ihi + ilo) >> 1;
00865
00866
00867
       if (xx[i] < xx[i + 1])
         while (ihi > ilo + 1) {
  i = (ihi + ilo) >> 1;
00868
00869
            if (xx[i] > x)
00870
00871
             ihi = i;
00872
           else
00873
             ilo = i;
00874
       } else
00875
         while (ihi > ilo + 1) {
           i = (ihi + ilo) >> 1;
00876
00877
           <u>if</u> (xx[i] <= x)
00878
            ihi = i;
00879
           else
00880
             ilo = i;
00881
         }
00882
00883
       return ilo;
00884 }
```

```
00887
00888 void read atm(
        const char *filename,
ctl_t * ctl,
00889
00890
        atm_t * atm) {
00892
00893
        FILE *in:
00894
        char line[LEN], *tok;
00895
00896
00897
         double t0;
00898
00899
        int dimid, ip, iq, ncid, varid;
00900
00901
         size_t nparts;
00902
00903
        /* Init... */
00904
         atm->np = 0;
00905
00906
         /* Write info... */
00907
         printf("Read atmospheric data: %s\n", filename);
00908
00909
         /* Read ASCII data... */
00910
         if (ctl->atm_type == 0) {
00911
00912
            /* Open file... */
           if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");
00913
00914
00915
00916
           /* Read line... */
00917
           while (fgets(line, LEN, in)) {
00918
             /* 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]);
00919
00920
00921
00923
00924
00925
00926
00927
              /\star Convert altitude to pressure... \star/
             atm->p[atm->np] = P(atm->p[atm->np]);
00928
00929
00930
              /* Increment data point counter... */
00931
              if ((++atm->np) > NP)
00932
                ERRMSG("Too many data points!");
00933
           }
00934
00935
            /* Close file... */
00936
           fclose(in);
00937
00938
00939
         /* Read binary data... */
00940
         else if (ctl->atm_type == 1) {
00941
00942
            /* Open file... */
           if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");
00943
00944
00945
00946
            /* Read data... */
00947
           FREAD(&atm->np, int,
00948
00949
                  in);
           FREAD(atm->time, double, (size_t) atm->np,
00950
00951
00952
                  in);
00953
           FREAD (atm->p, double,
00954
                    (size_t) atm->np,
00955
                  in);
00956
           FREAD(atm->lon, double,
00957
                    (size_t) atm->np,
                  in);
00958
           FREAD(atm->lat, double,
00959
00960
                    (size_t) atm->np,
00961
                  in);
00962
            for (iq = 0; iq < ctl->nq; iq++)
00963
             FREAD(atm->q[iq], double,
00964
                      (size_t) atm->np,
00965
                    in);
00966
            /* Close file... */
00967
00968
           fclose(in);
00969
00970
00971
        /* Read netCDF data... */
```

```
else if (ctl->atm_type == 2) {
00973
00974
           /* Open file... */
00975
           NC(nc_open(filename, NC_NOWRITE, &ncid));
00976
           /* Get dimensions... */
NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
00977
00979
           NC(nc_inq_dimlen(ncid, dimid, &nparts));
00980
           atm->np = (int) nparts;
00981
           if (atm->np > NP)
             ERRMSG("Too many particles!");
00982
00983
00984
           /* Get time... */
           NC(nc_inq_varid(ncid, "time", &varid));
00985
00986
           NC(nc_get_var_double(ncid, varid, &t0));
00987
           for (ip = 0; ip < atm->np; ip++)
00988
             atm->time[ip] = t0;
00989
           /* Read geolocations... */
NC(nc_inq_varid(ncid, "PRESS", &varid));
00991
           NC(nc_get_var_double(ncid, varid, atm->p));
NC(nc_inq_varid(ncid, "LON", &varid));
00992
00993
           NC(nc_get_var_double(ncid, varid, atm->lon));
NC(nc_inq_varid(ncid, "LAT", &varid));
00994
00995
00996
           NC(nc_get_var_double(ncid, varid, atm->lat));
00998
           /* Read variables... */
00999
           if (ctl->qnt_p >= 0)
             if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
01000
01001
           \label{eq:nc_qet_var_double} $$NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]))$; if (ctl->qnt_t >= 0)
01002
01003
             if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
01004
                NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
01005
           if (ctl->qnt_u >= 0)
           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)
01006
01007
01008
             if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
01010
                NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
01011
           if (ctl->qnt_w >= 0)
01012
              if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
               \label{eq:ncd_def} \mbox{NC(nc\_get\_var\_double(ncid, varid, atm->q[ctl->qnt\_w]));}
01013
01014
           if (ct.1->ant. h2o >= 0)
             if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
01015
01016
                NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
01017
           if (ctl->qnt_o3 >= 0)
01018
            if (nc_inq_varid(ncid, "03", &varid) == NC_NOERR)
01019
                \label{local_nc_delta} \mbox{NC(nc\_get\_var\_double(ncid, varid, atm->q[ctl->qnt\_o3]));}
01020
           if (ctl->qnt_theta >= 0)
             if (nc_ing_varid(ncid, "THETA", &varid) == NC_NOERR)
01021
                NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
01023
           if (ctl->qnt_pv >= 0)
              if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
01024
01025
               NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
01026
01027
           /* Check data... */
           for (ip = 0; ip < atm->np; ip++)
              if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
01029
01030
                  || (ct1->qnt_t >= 0 \&\& fabs(atm->q[ct1->qnt_t][ip]) > 350)
01031
                  \label{eq:ctl-parth20} \mbox{ | (ctl-part_h2o) = 0 && fabs(atm-part_h2o)[ip]) > 1)}
01032
                  | | (ctl->qnt\_theta >= 0 \&\& fabs(atm->q[ctl->qnt\_theta][ip]) > 1e10)
                  || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
01033
01034
                atm->time[ip] = GSL_NAN;
                atm->p[ip] = GSL_NAN;
01035
01036
                atm->lon[ip] = GSL_NAN;
01037
                atm->lat[ip] = GSL_NAN;
                for (iq = 0; iq < ctl->nq; iq++)
  atm->q[iq][ip] = GSL_NAN;
01038
01039
01040
             } else {
               if (ctl->qnt_h2o >= 0)
01042
                  atm->q[ctl->qnt_h2o][ip] *= 1.608;
01043
                if (ctl->qnt_pv >= 0)
                atm->q[ctl->qnt_pv][ip] *= le6;
if (atm->lon[ip] > 180)
01044
01045
01046
                  atm->lon[ip] -= 360;
01047
01048
01049
           /* Close file... */
01050
           NC(nc_close(ncid));
01051
01052
         /* Error... */
01054
01055
           ERRMSG("Atmospheric data type not supported!");
01056
01057
         /* Check number of points... */
01058
         if (atm->np < 1)</pre>
```

```
ERRMSG("Can not read any data!");
01060 }
01061
01063
01064 void read ctl(
      const char *filename,
01066
       int argc,
01067
       char *argv[],
01068
       ctl_t * ctl) {
01069
01070
       int ip, ia;
01071
01072
        /* Write info... */
01073
       printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
                "(executable: %s | compiled: %s, %s)\n\n",
01074
01075
               argv[0], __DATE__, __TIME__);
01076
       /* Initialize quantity indices... */
01078
       ctl->qnt_ens = -1;
01079
        ctl->qnt_m = -1;
        ctl->qnt_r = -1;
01080
       ctl->qnt_rho = -1;
01081
       ctl->qnt_ps = -1;
01082
01083
        ctl->qnt_pt = -1;
       ctl->qnt_z = -1;
01084
01085
        ctl->qnt_p = -1;
01086
       ctl->qnt_t = -1;
       ctl->qnt_u = -1;
01087
       ctl->qnt_v = -1;
01088
01089
       ctl->qnt_w = -1;
01090
        ctl->qnt\_h2o = -1;
01091
        ct1->qnt_o3 = -1;
01092
        ctl->qnt\_theta = -1;
01093
        ctl->qnt_vh = -1;
        ctl->qnt_vz = -1;
01094
       ctl->qnt_pv = -1;
01095
       ctl->qnt\_tice = -1;
01097
        ctl->qnt_tsts = -1;
01098
        ctl \rightarrow qnt_tnat = -1;
01099
        ctl->qnt\_stat = -1;
01100
       /* Read quantities... */
01101
01102
        ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
        if (ctl->nq > NQ)
01103
01104
          ERRMSG("Too many quantities!");
01105
        for (iq = 0; iq < ctl->nq; iq++) {
01106
          /* Read quantity name and format... */
01107
          scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
01108
01109
                   ctl->qnt_format[iq]);
01110
01111
          /* Try to identify quantity... */
if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
01112
01113
           ctl->qnt_ens = iq;
01114
            sprintf(ctl->qnt_unit[iq], "-");
01116
          } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
01117
            ctl->qnt_m = iq;
01118
            sprintf(ctl->qnt_unit[iq], "kg");
          } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
  ctl->qnt_r = iq;
01119
01120
01121
            sprintf(ctl->qnt_unit[iq], "m");
01122
          } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
            ctl->qnt_rho = iq;
01123
01124
           sprintf(ctl->qnt_unit[iq], "kg/m^3");
          } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
ctl->qnt_ps = iq;
01125
01126
            sprintf(ctl->qnt_unit[iq], "hPa");
01127
01128
          } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
01129
           ctl->qnt_pt = iq;
01130
            sprintf(ctl->qnt_unit[iq], "hPa");
          } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
01131
           ctl->qnt_z = iq;
sprintf(ctl->qnt_unit[iq], "km");
01132
01133
01134
          } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
            ctl->qnt_p = iq;
01135
01136
            sprintf(ctl->qnt_unit[iq], "hPa");
          } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
  ctl->qnt_t = iq;
  sprintf(ctl->qnt_unit[iq], "K");
01137
01138
01139
01140
          } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
            ctl->qnt_u = iq;
01141
01142
            sprintf(ctl->qnt_unit[iq], "m/s");
          } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
01143
           ctl->qnt_v = iq;
sprintf(ctl->qnt_unit[iq], "m/s");
01144
01145
```

```
} else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
             ctl->qnt_w = iq;
01148
              sprintf(ctl->qnt_unit[iq], "hPa/s");
            } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
01149
01150
              ctl->qnt_h2o = iq;
              sprintf(ctl->qnt_unit[iq], "1");
01151
01152
            } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
01153
              ctl->qnt\_o3 = iq;
01154
              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");
01155
01156
01157
           } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
ctl->qnt_vh = iq;
01158
01159
01160
              sprintf(ctl->qnt_unit[iq], "m/s");
01161
            } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
01162
              ctl->qnt_vz = iq;
              sprintf(ctl->qnt_unit[iq], "m/s");
01163
01164
            } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
              ctl->qnt_pv = iq;
              sprintf(ctl->qnt_unit[iq], "PVU");
01166
           } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
  ctl->qnt_tice = iq;
  sprintf(ctl->qnt_unit[iq], "K");
01167
01168
01169
           } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
01170
             ctl->qnt_tsts = iq;
01171
              sprintf(ctl->qnt_unit[iq], "K");
01172
01173
           } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
01174
              ctl->qnt_tnat = iq;
              sprintf(ctl->qnt_unit[iq], "K");
01175
           } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
01176
              ctl->qnt_stat = iq;
01177
01178
              sprintf(ctl->qnt_unit[iq], "-");
01179
01180
              scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
01181
01182
01183
          /* Time steps of simulation... */
01184
         ctl->direction :
           (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
01185
           f (ctl->direction != -1 && ctl->direction != 1)
ERRMSG("Set DIRECTION to -1 or 1!");
01186
01187
         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);
01188
01189
01190
01191
          /* Meteorological data..
         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);
01192
01193
01194
         ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
01195
         ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
01196
         ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "l", NULL); ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "l", NULL); ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
01197
01198
01199
         if (ctl->met_np > EP)
01200
           ERRMSG("Too many levels!");
01201
         for (ip = 0; ip < ctl->met_np; ip++)
           ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
01203
         ctl->met_tropo
01204
         = (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);
01205
01206
01207
01208
01209
          /* Isosurface parameters... */
01210
         ctl->isosurf
         = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
01211
01212
01213
01214
         /* Diffusion parameters... */
         ctl->turb_dx_trop
           = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
01216
01217
         ctl->turb_dx_strat
01218
            = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
01219
         ctl->turb dz trop
01220
            = scan ctl(filename, argc, argv, "TURB DZ TROP", -1, "0", NULL);
01221
         ctl->turb dz strat
01222
            = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
01223
         ctl->turb_mesox =
            scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
01224
01225
         ctl->turb mesoz =
            scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
01226
01228
          /* Mass and life time..
01229
         ctl->molmass = scan_ctl(filename, argc, argv, "MOLMASS", -1, "1", NULL);
01230
         ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
         ctl->tdec strat =
01231
01232
            scan ctl(filename, argv, "TDEC STRAT", -1, "0", NULL);
```

```
01233
01234
         /* PSC analysis... */
01235
         ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
         ctl->psc hno3 =
01236
           scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
01237
01238
         /* Output of atmospheric data... */
scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
01239
01240
      atm_basename);
01241
         scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
01242
         ctl->atm_dt_out =
           scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
01243
01244
         ctl->atm_filter
            (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
01245
01246
         ctl->atm_type =
01247
           (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
01248
01249
         /* Output of CSI data... */
01250
         scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
01251 ctl->csi_dt_out =
        scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->
01252
01253
      csi obsfile):
01254
        ctl->csi_obsmin =
01255
           scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
01256
         ctl->csi_modmin =
         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);
01257
01258
01259
01260
01261
         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);
01262
01263
01264
         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);
01265
                                                                                       -90". NUITI):
01266
01267
01268
         ctl->csi_ny =
01269
           (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
01270
01271
         /* Output of ensemble data... */
         scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
01272
      ens_basename);
01273
01274
          /* Output of grid data... */
01275
         scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
01276
                    ctl->grid_basename);
         scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
01277
grid_gpfile);
01278 ct1->~
        ctl->grid_dt_out =
01279
            scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
01280
         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);
01281
01282
01283
         ctl->grid_nz =
01284
01285
            (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
01286
         ctl->grid_lon0 =
01287
           scan_ctl(filename, argc, argv, "GRID_LONO", -1, "-180", NULL);
01288
         ct1->grid lon1 =
01289
           scan ctl(filename, argc, argv, "GRID LON1", -1, "180", NULL);
01290
         ctl->grid_nx =
            (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
01291
01292
         ctl->grid_lat0 =
01293
           scan_ctl(filename, argc, argv, "GRID_LATO", -1, "-90", NULL);
01294
         ctl->grid lat1
01295
           scan ctl(filename, argc, argv, "GRID LAT1", -1, "90", NULL);
01296
         ctl->grid_ny =
01297
            (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
01298
01299
         /* Output of profile data... */
         scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
01300
                    ctl->prof basename);
01301
         scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
01302
      prof_obsfile);
01303
         ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
01304
         ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
         ctl->prof nz =
01305
01306
           (int) scan ctl(filename, argc, argv, "PROF NZ", -1, "60", NULL);
01307
         ctl->prof lon0 =
01308
           scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
01309
         ctl->prof lon1
01310
           scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
         ctl->prof_nx =
01311
           (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
01312
01313
         ctl->prof_lat0 =
```

```
01314
           scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
01315
01316
          scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
01317
        ctl->prof_ny =
          (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
01318
01319
01320
        /* Output of station data... */
        scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
01321
01322
                  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);
01323
01324
01325
01326 }
01327
01329
01330 void read_met(
        ctl_t * ctl,
char *filename,
01331
01332
01333
        met_t * met) {
01334
01335
        char cmd[2 * LEN], levname[LEN], tstr[10];
01336
01337
        static float help[EX * EY];
01338
01339
        int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
01340
01341
        size_t np, nx, ny;
01342
01343
        /* Write info... */
01344
        printf("Read meteorological data: %s\n", filename);
01345
01346
         /* Get time from filename... */
01347
        sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
01348
        year = atoi(tstr);
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
01349
01350
        mon = atoi(tstr);
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
01351
01352
        day = atoi(tstr);
01353
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
01354
        hour = atoi(tstr);
01355
        time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
01356
01357
        /* Open netCDF file... */
        if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
01358
01359
01360
           /* Try to stage meteo file... */
          01361
01362
01363
             if (system(cmd) != 0)
01364
01365
               ERRMSG("Error while staging meteo data!");
01366
01367
           /* Try to open again... */
01368
          NC(nc_open(filename, NC_NOWRITE, &ncid));
01369
01370
01371
         /* Get dimensions... */
01372
        NC(nc_inq_dimid(ncid, "lon", &dimid));
NC(nc_inq_dimlen(ncid, dimid, &nx));
if (nx < 2 || nx > EX)
01373
01374
01375
01376
          ERRMSG("Number of longitudes out of range!");
01377
        NC(nc_inq_dimid(ncid, "lat", &dimid));
01378
01379
        NC(nc_inq_dimlen(ncid, dimid, &ny));
01380
            (ny < 2 \mid \mid ny > EY)
          ERRMSG("Number of latitudes out of range!");
01381
01382
01383
        sprintf(levname, "lev");
01384
        NC(nc_inq_dimid(ncid, levname, &dimid));
01385
        NC(nc_inq_dimlen(ncid, dimid, &np));
        if (np == 1) {
   sprintf(levname, "lev_2");
01386
01387
          NC(nc_inq_dimid(ncid, levname, &dimid));
NC(nc_inq_dimlen(ncid, dimid, &np));
01388
01389
01390
        if (np < 2 || np > EP)
   ERRMSG("Number of levels out of range!");
01391
01392
01393
01394
        /* Store dimensions... */
        met->np = (int) np;
met->nx = (int) nx;
01395
01396
01397
        met->ny = (int) ny;
01398
        /* Get horizontal grid... */
NC(nc_ing_varid(ncid, "lon", &varid));
01399
01400
```

```
NC(nc_get_var_double(ncid, varid, met->lon));
         NC(nc_inq_varid(ncid, "lat", &varid));
01402
01403
         NC(nc_get_var_double(ncid, varid, met->lat));
01404
         /* 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, "w", "V", met, met->v, 0.01f);
read_met_help(ncid, "q", "Q", met, met->b(float) (MA / 18.01528));
read_met_help(ncid, "o3", "03", met, met->o3, (float) (MA / 48.00));
01405
01406
01407
01408
01409
01410
01411
01412
01413
         /* Meteo data on pressure levels... */
01414
         if (ctl->met_np <= 0) {</pre>
01415
01416
            /* Read pressure levels from file... */
01417
           NC(nc_inq_varid(ncid, levname, &varid));
           NC(nc_get_var_double(ncid, varid, met->p));
for (ip = 0; ip < met->np; ip++)
01418
01419
              met->p[ip] /= 100.;
01420
01421
01422
            /* Extrapolate data for lower boundary... */
01423
           read_met_extrapolate(met);
01424
01425
01426
         /* Meteo data on model levels... */
01427
01428
            /* Read pressure data from file... */
read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
01429
01430
01431
01432
            /\star Interpolate from model levels to pressure levels... \star/
01433
            read_met_ml2pl(ctl, met, met->t);
01434
            read_met_ml2pl(ctl, met, met->u);
01435
            read_met_ml2pl(ctl, met, met->v);
01436
            read_met_ml2pl(ctl, met, met->w);
            read_met_ml2pl(ctl, met, met->h2o);
01437
01438
            read_met_ml2pl(ctl, met, met->o3);
01439
01440
            /* Set pressure levels... */
           met->np = ctl->met_np;
for (ip = 0; ip < met->np; ip++)
01441
01442
              met->p[ip] = ctl->met_p[ip];
01443
01444
01445
01446
          /* Check ordering of pressure levels... */
         for (ip = 1; ip < met->np; ip++)
  if (met->p[ip - 1] < met->p[ip])
01447
01448
              ERRMSG("Pressure levels must be descending!");
01449
01450
01451
          /* Read surface pressure... */
         01452
01453
01454
            NC(nc_get_var_float(ncid, varid, help));
         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
01455
01456
01458
01459
                       || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
01460
            NC(nc_get_var_float(ncid, varid, help));
01461
            for (iy = 0; iy < met->ny; iy++)
  for (ix = 0; ix < met->nx; ix++)
01462
01463
                met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
01464
         } else
01465
            for (ix = 0; ix < met->nx; ix++)
01466
              for (iy = 0; iy < met\rightarrowny; iy++)
01467
                 met->ps[ix][iy] = met->p[0];
01468
01469
         /* Create periodic boundary conditions... */
01470
         read_met_periodic(met);
01471
01472
         /* Calculate geopotential heights... */
01473
         read_met_geopot(ctl, met);
01474
01475
         /* Calculate potential vorticity... */
01476
         read_met_pv(met);
01477
01478
         /* Calculate tropopause pressure... */
01479
         read_met_tropo(ctl, met);
01480
         /* Downsampling... */
01481
01482
         read_met_sample(ctl, met);
01483
01484
          /* Close file... */
01485
        NC(nc_close(ncid));
01486 }
01487
```

```
01489
01490 void read_met_extrapolate(
01491
       met_t * met) {
01492
01493
        int ip, ip0, ix, iv;
01494
01495
        /* Loop over columns... */
01496
        for (ix = 0; ix < met->nx; ix++)
01497
          for (iy = 0; iy < met->ny; iy++) {
01498
01499
             /* Find lowest valid data point... */
            for (ip0 = met->np - 1; ip0 >= 0; ip0--)
    if (!gsl_finite(met->t[ix][iy][ip0])
01500
01501
01502
                   || !gsl_finite(met->u[ix][iy][ip0])
01503
                   || !gsl_finite(met->v[ix][iy][ip0])
01504
                   || !gsl_finite(met->w[ix][iy][ip0]))
01505
                break;
01506
             /* Extrapolate... */
01508
            for (ip = ip0; ip >= 0; ip--)
01509
              met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
              met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
01510
              met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
01511
01512
              met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
              met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
01513
01514
              met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
01515
01516
01517 }
01518
01520
01521 void read_met_geopot(
        ctl_t * ctl,
met_t * met) {
01522
01523
01524
01525
        static double topo_lat[EY], topo_lon[EX], topo_z[EX][EY];
01526
01527
        static int init, topo_nx = -1, topo_ny;
01528
01529
        FILE *in:
01530
01531
        char line[LEN];
01532
01533
        double data[30], lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;
01534
01535
        float help[EX][EY];
01536
        int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;
01537
01538
01539
         /* Initialize geopotential heights... */
01540
        for (ix = 0; ix < met->nx; ix++)
         for (iy = 0; iy < met->ny; iy++)
  for (ip = 0; ip < met->np; ip++)
01541
01542
              met->z[ix][iy][ip] = GSL_NAN;
01543
01544
01545
        /* Check filename... */
01546
       if (ctl->met_geopot[0] == '-')
01547
          return;
01548
01549
        /* Read surface geopotential... */
01550
        if (!init) {
01551
         init = 1;
01552
01553
          /* Write info... */
          printf("Read surface geopotential: %s\n", ctl->met_geopot);
01554
01555
01556
          /* Open file... */
          if (!(in = fopen(ctl->met_geopot, "r")))
    ERRMSG("Cannot open file!");
01558
01559
          /* Read data... */
while (fgets(line, LEN, in))
  if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rz) == 3) {
01560
01561
01562
01563
              if (rlon != rlon_old) {
01564
                <u>if</u> ((++topo_nx) >= EX)
01565
                  ERRMSG("Too many longitudes!");
01566
                topo_ny = 0;
              }
01567
              rlon_old = rlon;
01568
01569
              topo_lon[topo_nx] = rlon;
01570
               topo_lat[topo_ny] = rlat;
01571
               topo_z[topo_nx][topo_ny] = rz;
              if ((++topo_ny) >= EY)
   ERRMSG("Too many latitudes!");
01572
01573
01574
            }
```

```
01575
         if ((++topo_nx) >= EX)
          ERRMSG("Too many longitudes!");
01576
01577
01578
         /* Close file... */
01579
         fclose(in);
01580
01581
         /* Check grid spacing... */
01582
         || fabs(met->lat[0] - met->lat[1]) != fabs(topo_lat[0] - topo_lat[1]))
01583
01584
           printf("Warning: Grid spacing does not match!\n");
01585
01586
01587
       /* Apply hydrostatic equation to calculate geopotential heights... */
01588
       for (ix = 0; ix < met->nx; ix++)
01589
         for (iy = 0; iy < met->ny; iy++) {
01590
           /* Get surface height... */
01591
01592
           lon = met->lon[ix];
           if (lon < topo_lon[0])</pre>
01593
01594
             lon += 360;
01595
           else if (lon > topo_lon[topo_nx - 1])
01596
            lon -= 360;
           lat = met->lat[iy];
01597
          01598
01599
01600
01601
           01602
01603
           z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);
01604
01605
01606
           /* Find surface pressure level... */
01607
           ip0 = locate(met->p, met->np, met->ps[ix][iy]);
01608
           01609
01610
01611
01612
01613
           /* Upper part of profile... */
           met->z[ix][iy][ip0 + 1]
= (float) (z0 + RI / MA / G0 * 0.5 * (ts + met->t[ix][iy][ip0 + 1])
01614
01615
                       * log(met->ps[ix][iy] / met->p[ip0 + 1]));
01616
           for (ip = ip0 + 2; ip < met->np; ip++)
01617
01618
            met->z[ix][iy][ip]
01619
              = (float) (met->z[ix][iy][ip - 1] + RI / MA / G0
                         * 0.5 * (met->t[ix][iy][ip - 1] + met->t[ix][iy][ip])
01620
01621
                         * log(met->p[ip - 1] / met->p[ip]));
01622
         }
01623
01624
       /* Smooth fields... */
01625
       for (ip = 0; ip < met->np; ip++) {
01626
01627
         /* Median filter... */
01628
         for (ix = 0; ix < met->nx; ix++)
           for (iy = 0; iy < met->nx; iy++) {
01629
01630
            n = 0;
             for (ix2 = ix - 2; ix2 <= ix + 2; ix2++) {</pre>
01631
              ix3 = ix2;
01632
01633
              if (ix3 < 0)
01634
                ix3 += met->nx;
               if (ix3 \ge met - \ge nx)
01635
                ix3 -= met->nx;
01636
01637
               for (iy2 = GSL_MAX(iy - 2, 0); iy2 <= GSL_MIN(iy + 2, met->ny - 1);
                   iy2++)
01638
01639
                if (gsl_finite(met->z[ix3][iy2][ip])) {
01640
                 data[n] = met \rightarrow z[ix3][iy2][ip];
01641
                  n++;
                }
01642
01643
01644
             if (n > 0) {
               gsl_sort(data, 1, (size_t) n);
help[ix][iy] = (float)
01645
01646
01647
                gsl_stats_median_from_sorted_data(data, 1, (size_t) n);
01648
              help[ix][iy] = GSL_NAN;
01649
01650
01651
         /* Copy data... */
for (ix = 0; ix < met->nx; ix++)
for (iy = 0; iy < met->nx; iy++)
met->z[ix][iy][ip] = help[ix][iy];
01652
01653
01654
01655
01656
01657 }
01658
01660
01661 void read met help(
```

```
01662
       int ncid,
       char *varname,
01663
01664
       char *varname2,
01665
       met_t * met,
01666
       float dest[EX][EY][EP],
01667
       float scl) {
01668
01669
       static float help[EX * EY * EP];
01670
01671
       int ip, ix, iy, varid;
01672
       /* Check if variable exists... */
if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
01673
01674
01675
         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
01676
01677
       /* Read data... */
NC(nc_get_var_float(ncid, varid, help));
01678
01679
01680
        /* Copy and check data... */
01682
       for (ix = 0; ix < met->nx; ix++)
01683
         for (iy = 0; iy < met->ny; iy++)
           for (ip = 0; ip < met >ny, ip++) {
  dest[ix][iy][ip] = help[(ip * met ->ny + iy) * met ->nx + ix];
  if (fabsf(dest[ix][iy][ip]) < le14f)
   dest[ix][iy][ip] *= scl;</pre>
01684
01685
01686
01687
01688
              else
01689
               dest[ix][iy][ip] = GSL_NAN;
01690
01691 }
01692
01694
01695 void read_met_ml2pl(
       ctl_t * ctl,
met_t * met,
01696
01697
01698
       float var[EX][EY][EP]) {
01699
01700
       double aux[EP], p[EP], pt;
01701
01702
       int ip, ip2, ix, iy;
01703
01704
       /* Loop over columns... */
01705
       for (ix = 0; ix < met->nx; ix++)
01706
         for (iy = 0; iy < met->ny; iy++) {
01707
01708
            /* Copy pressure profile... */
           for (ip = 0; ip < met->np; ip++)
  p[ip] = met->pl[ix][iy][ip];
01709
01710
01711
            /* Interpolate... */
for (ip = 0; ip < ctl->met_np; ip++) {
01713
             pt = ctl->met_p[ip];
01714
01715
              01716
               pt = p[0];
             01717
01718
01719
01720
             ip2 = locate(p, met->np, pt);
             01721
01722
01723
01724
01725
            /* Copy data... */
01726
            for (ip = 0; ip < ctl->met_np; ip++)
01727
             var[ix][iy][ip] = (float) aux[ip];
01728
01729 }
01730
01732
01733 void read_met_periodic(
01734
       met_t * met) {
01735
01736
       int ip, iv;
01737
01738
       /* Check longitudes... */
01739
       if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
                   + met - > lon[1] - met - > lon[0] - 360) < 0.01))
01740
01741
         return:
01742
01743
       /* Increase longitude counter... */
       if ((++met->nx) > EX)
01744
01745
         ERRMSG("Cannot create periodic boundary conditions!");
01746
       /* Set longitude... */
met->lon[met->nx - 2] + met->lon[1] - met->
01747
01748
```

```
lon[0];
01749
01750
          /* Loop over latitudes and pressure levels... */
01751
          for (iy = 0; iy < met->ny; iy++)
           for (ip = 0; ip < met->np; ip++) {
  met->ps[met->nx - 1][iy] = met->ps[0][iy];
  met->pt[met->nx - 1][iy] = met->pt[0][iy];
01752
01753
01754
              met->p[met->nx - 1][iy] = met->p[0][iy][ip];
met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
met->t[met->nx - 1][iy][ip] = met->u[0][iy][ip];
met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
01755
01756
01757
01758
              met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
01759
              met->pv[met->nx - 1][iy][ip] = met->pv[0][iy][ip];
01760
01761
              met \rightarrow h2o[met \rightarrow nx - 1][iy][ip] = met \rightarrow h2o[0][iy][ip];
01762
              met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01763
01764 }
01765
01767
01768 void read_met_pv(
01769
         met_t * met) {
01770
01771
         double c0, c1, cr, dx, dy, dp, dtdx, dvdx, dtdy, dudy, dtdp, dudp, dvdp,
latr, vort, pows[EP];
01772
01773
         int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
01774
01775
01776
         /* Set powers... */
         for (ip = 0; ip < met->np; ip++)
01777
01778
           pows[ip] = pow(1000. / met->p[ip], 0.286);
01779
01780
          /* Loop over grid points... */
01781
         for (ix = 0; ix < met->nx; ix++) {
01782
           /* Set indices... */
ix0 = GSL_MAX(ix - 1, 0);
01783
01784
            ix1 = GSL_MIN(ix + 1, met -> nx - 1);
01785
01786
01787
            /* Loop over grid points... */
01788
            for (iy = 0; iy < met->ny; iy++) {
01789
01790
              /* Set indices... */
iy0 = GSL_MAX(iy - 1, 0);
01791
01792
              iy1 = GSL_MIN(iy + 1, met -> ny - 1);
01793
01794
               /* Set auxiliary variables... */
              latr = GSL_MIN(GSL_MAX(met->lat[iy], -89.), 89.);
01795
              dx = 1000. * deg2dx(met->lon[ix1] - met->lon[ix0], latr);
dy = 1000. * deg2dy(met->lat[iy1] - met->lat[iy0]);
01796
01797
              c0 = cos(met->lat[iy0] / 180. * M_PI);
c1 = cos(met->lat[iy1] / 180. * M_PI);
01798
01799
01800
               cr = cos(latr / 180. * M_PI);
01801
              vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
01802
01803
              /* Loop over grid points... */
for (ip = 0; ip < met->np; ip++) {
01805
                 /* Set indices... */
ip0 = GSL_MAX(ip - 1, 0);
01806
01807
                 ip1 = GSL_MIN(ip + 1, met->np - 1);
01808
01809
01810
                 /* Set auxiliary variables... */
                 dp = 100. * (met -> p[ip1] - met -> p[ip0]);
01811
01812
01813
                 /* Get gradients in longitude... */
                 01814
01815
01816
                 /* 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;
01818
01819
01820
                 /* Get gradients in pressure... */
01821
01822
                 dtdp =
                  (met->t[ix][iy][ip1] * pows[ip1] -
01823
01824
                     met->t[ix][iy][ip0] * pows[ip0]) / dp;
                 dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / dp;
dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / dp;
01825
01826
01827
                 /* Calculate PV... */
01828
01829
                 met \rightarrow pv[ix][iy][ip] = (float)
01830
01831
                     (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
01832
01834 }
```

```
01836
01837 /
         ******************************
01838
01839 void read_met_sample(
01840
          ctl_t * ctl,
          met_t * met) {
01841
01842
01843
          met_t *help;
01844
01845
          float w, wsum;
01846
01847
          int ip, ip2, ix, ix2, iv, iv2;
01848
01849
          /* Check parameters... */
01850
          if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1)</pre>
01851
             return:
01852
01853
           /* Allocate... */
01854
          ALLOC(help, met_t, 1);
01855
01856
           /* Copy data... */
          help->nx = met->nx;
help->ny = met->ny;
01857
01858
01859
          help->np = met->np;
01860
          memcpy(help->lon, met->lon, sizeof(met->lon));
          memcpy(help->lat, met->lat, sizeof(met->lat));
01861
01862
          memcpy(help->p, met->p, sizeof(met->p));
01863
01864
          /* Smoothing... */
01865
          for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
01866
             for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
01867
                for (ip = 0; ip < met->np; ip += ctl->met_dp) {
01868
                  help->ps[ix][iy] = 0;
                  help \rightarrow pt[ix][iy] = 0;
01869
01870
                  help \rightarrow z[ix][iy][ip] = 0;
                  help->t[ix][iy][ip] = 0;
help->u[ix][iy][ip] = 0;
01871
01872
01873
                  help \rightarrow v[ix][iy][ip] = 0;
01874
                   help \rightarrow w[ix][iy][ip] = 0;
01875
                  help \rightarrow pv[ix][iy][ip] = 0;
                  help->h2o[ix][iy][ip] = 0;
01876
01877
                  help \rightarrow 03[ix][iy][ip] = 0;
01878
                  wsum = 0;
                  for (ix2 = GSL_MAX(ix - ctl->met_sx + 1, 0);
01879
                     ix2 <= GSL_MIN(ix + ctl->met_sx - 1, met->nx - 1); ix2++)
for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
01880
01881
                       iv2 = GSL_MAX(iy - cti->met_sy + 1, 0);
iv2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iv2++)
for (ip2 = GSL_MIX(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)
    * (float) (1.0 - fabs(ip - ip2) / ctl->met_sp);
}
01882
01883
01884
01885
01886
01887
                          help->ps[ix][iy] += w * met->ps[ix2][iy2];
help->pt[ix][iy] += w * met->pt[ix2][iy2];
01888
01889
                          help->z[ix][iy][ip] += w * met->z[ix2][iy2][ip2];
help->t[ix][iy][ip] += w * met->t[ix2][iy2][ip2];
01890
01892
                          help \rightarrow u[ix][iy][ip] += w * met \rightarrow u[ix2][iy2][ip2];
01893
                          help \rightarrow v[ix][iy][ip] += w * met \rightarrow v[ix2][iy2][ip2];
01894
                          help \rightarrow w[ix][iy][ip] += w * met \rightarrow w[ix2][iy2][ip2];
                          help->pv[ix][iy][ip] += w * met->pv[ix2][iy2][ip2];
help->h2o[ix][iy][ip] += w * met->h2o[ix2][iy2][ip2];
help->o3[ix][iy][ip] += w * met->o3[ix2][iy2][ip2];
01895
01896
01897
01898
                          wsum += w;
01899
01900
                  help->ps[ix][iy] /= wsum;
                  help->pt[ix][iy] /= wsum;
01901
01902
                   help->t[ix][iy][ip] /= wsum;
                  help->z[ix][iy][ip] /= wsum;
help->u[ix][iy][ip] /= wsum;
01903
01904
01905
                   help->v[ix][iy][ip] /= wsum;
01906
                  help->w[ix][iy][ip] /= wsum;
                  help->pv[ix][iy][ip] /= wsum;
help->h2o[ix][iy][ip] /= wsum;
01907
01908
                  help->o3[ix][iy][ip] /= wsum;
01909
01910
01911
             }
01912
01913
          /* Downsampling... */
01914
01915
          met->nx = 0;
          for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
01916
01917
            met->lon[met->nx] = help->lon[ix];
01918
             met->ny = 0;
01919
             for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
               met->lat[met->ny] = help->lat[iy];
01920
01921
               met->ps[met->nx][met->ny] = help->ps[ix][iy];
```

```
met->pt[met->nx][met->ny] = help->pt[ix][iy];
            met->np = 0;
for (ip = 0; ip < help->np; ip += ctl->met_dp) {
01923
01924
01925
              met->p[met->np] = help->p[ip];
01926
               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];
01927
               met->u[met->nx] [met->ny] [met->np] = help->u[ix][iy][ip];
01928
01929
               met \rightarrow v[met \rightarrow nx][met \rightarrow ny][met \rightarrow np] = help \rightarrow v[ix][iy][ip];
               met->w[met->nx] [met->ny] [met->np] = help->w[ix][iy][ip];
met->pv[met->nx] [met->ny] [met->np] = help->pv[ix][iy][ip];
01930
01931
               met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
01932
01933
               met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
01934
               met->np++;
01935
01936
             met->ny++;
01937
          met->nx++;
01938
        }
01939
01940
01941
         /* Free... */
01942
        free(help);
01943 }
01944
01946
01947 void read_met_tropo(
        ctl_t * ctl,
01948
01949
        met_t * met) {
01950
01951
        gsl_interp_accel *acc;
01952
01953
        gsl spline *spline;
01954
01955
        double p2[400], pv[400], pv2[400], t[400], t2[400], th[400], th2[400],
01956
         z[400], z2[400];
01957
01958
        int found, ix, iy, iz, iz2;
01959
01960
        /* Allocate... */
01961
        acc = gsl_interp_accel_alloc();
01962
        spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) met->np);
01963
01964
        /* Get altitude and pressure profiles... */
01965
        for (iz = 0; iz < met->np; iz++)
01966
          z[iz] = Z(met->p[iz]);
01967
        for (iz = 0; iz <= 170; iz++) {</pre>
01968
         z2[iz] = 4.5 + 0.1 * iz;
          p2[iz] = P(z2[iz]);
01969
01970
01971
01972
        /* Do not calculate tropopause... */
01973
        if (ctl->met_tropo == 0)
01974
          for (ix = 0; ix < met->nx; ix++)
01975
            for (iy = 0; iy < met->ny; iy++)
01976
               met->pt[ix][iy] = GSL_NAN;
01977
01978
        /* Use tropopause climatology... */
01979
        else if (ctl->met_tropo == 1)
01980
         for (ix = 0; ix < met->nx; ix++)
01981
             for (iy = 0; iy < met->ny; iy++)
01982
               met->pt[ix][iy] = clim_tropo(met->time, met->lat[iy]);
01983
01984
        /* Use cold point... */
01985
        else if (ctl->met_tropo == 2) {
01986
          /* Loop over grid points... */
for (ix = 0; ix < met->nx; ix++)
for (iy = 0; iy < met->ny; iy++) {
01987
01988
01989
01990
               /* Interpolate temperature profile... */
               for (iz = 0; iz < met->np; iz++)
  t[iz] = met->t[ix][iy][iz];
01992
01993
               gsl_spline_init(spline, z, t, (size_t) met->np);
for (iz = 0; iz <= 170; iz++)</pre>
01994
01995
                 t2[iz] = gsl_spline_eval(spline, z2[iz], acc);
01996
01997
01998
               /* Find minimum... */
               iz = (int) gsl_stats_min_index(t2, 1, 171);
if (iz <= 0 || iz >= 170)
01999
02000
                 met->pt[ix][iy] = GSL_NAN;
02001
02002
               else
02003
                 met \rightarrow pt[ix][iy] = p2[iz];
02004
02005
        }
02006
02007
        /* Use WMO definition... */
02008
        else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
```

```
02010
            /* Loop over grid points... */
02011
           for (ix = 0; ix < met->nx; ix++)
             for (iy = 0; iy < met->ny; iy++) {
02012
02013
                 /* Interpolate temperature profile... */
02014
                for (iz = 0; iz < met->np; iz++)
02016
                  t[iz] = met->t[ix][iy][iz];
                gsl_spline_init(spline, z, t, (size_t) met->np);
for (iz = 0; iz <= 160; iz++)</pre>
02017
02018
                  t2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02019
02020
02021
                 /* Find 1st tropopause... */
02022
                met->pt[ix][iy] = GSL_NAN;
02023
                 for (iz = 0; iz <= 140; iz++) {
                   found = 1;
02024
                   for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
  if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
      / log(p2[iz2] / p2[iz]) > 2.0) {
02025
02026
02028
                       found = 0;
02029
02030
                  if (found) {
  if (iz > 0 && iz < 140)</pre>
02031
02032
02033
                       met->pt[ix][iy] = p2[iz];
02034
                     break;
02035
02036
02037
                 /\star Find 2nd tropopause... \star/
02038
                if (ctl->met_tropo == 4) {
  met->pt[ix][iy] = GSL_NAN;
02039
02040
02041
                   for (; iz <= 140; iz++) {
02042
                     found = 1;
                     for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
  if (1000. * G0 / RA * log(t2[iz2] / t2[iz])</pre>
02043
02044
                            / log(p2[iz2] / p2[iz]) < 3.0) {
02045
                          found = 0;
02047
                          break;
02048
02049
                     if (found)
02050
                       break;
02051
02052
                   for (; iz <= 140; iz++) {
                     found = 1;
02053
02054
                     for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
                       02055
02056
                          found = 0;
02057
02058
                          break:
02060
                     if (found) {
02061
                       if (iz > 0 && iz < 140)
02062
                         met \rightarrow pt[ix][iy] = p2[iz];
02063
                       break:
02064
                     }
02065
02066
                }
02067
02068
02069
02070
         /* Use dynamical tropopause... */
         else if (ctl->met_tropo == 5) {
02072
02073
            /* Loop over grid points... */
           for (ix = 0; ix < met->nx; ix++)
02074
02075
              for (iy = 0; iy < met->ny; iy++) {
02076
                /* Interpolate potential vorticity profile... */
for (iz = 0; iz < met->np; iz++)
  pv[iz] = met->pv[ix][iy][iz];
02077
02079
02080
                 gsl_spline_init(spline, z, pv, (size_t) met->np);
02081
                 for (iz = 0; iz \leq 160; iz++)
02082
                  pv2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02083
02084
                 /* Interpolate potential temperature profile... */
02085
                for (iz = 0; iz < met->np; iz++)
02086
                  th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
                gsl_spline_init(spline, z, th, (size_t) met->np);
for (iz = 0; iz <= 160; iz++)</pre>
02087
02088
02089
                  th2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02091
                 /* Find dynamical tropopause 3.5 PVU + 380 K */
02092
                met->pt[ix][iy] = GSL_NAN;
                for (iz = 0; iz <= 160; iz++)
  if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
   if (iz > 0 && iz < 160)</pre>
02093
02094
02095
```

```
met \rightarrow pt[ix][iy] = p2[iz];
02097
                 break;
02098
                }
02099
            }
02100
        }
02101
02102
02103
          ERRMSG("Cannot calculate tropopause!");
02104
02105
02106
        gsl_spline_free(spline);
02107
        gsl_interp_accel_free(acc);
02108 }
02109
02111
02112 double scan_ctl(
02113
        const char *filename,
02114
        int argc,
02115
        char *argv[],
        const char *varname,
02116
02117
        int arridx,
        const char *defvalue,
02118
02119
        char *value) {
02120
02121
        FILE *in = NULL;
02122
02123
        char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
         msg[2 * LEN], rvarname[LEN], rval[LEN];
02124
02125
02126
        int contain = 0, i;
02127
02128
        /* Open file... */
02129
        if (filename[strlen(filename) - 1] != '-')
         if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");
02130
02131
02132
02133
        /* Set full variable name... */
02134
        if (arridx >= 0) {
        sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
02135
02136
02137
        } else {
         sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
02138
02139
02140
02141
02142
        /* Read data... */
02143
        if (in != NULL)
          while (fgets(line, LEN, in))
  if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
  if (strcasecmp(rvarname, fullname1) == 0 | |
02144
02145
02146
02147
                  strcasecmp(rvarname, fullname2) == 0) {
02148
                contain = 1;
02149
                break;
02150
        for (i = 1; i < argc - 1; i++)</pre>
02151
         if (strcasecmp(argv[i], fullname1) == 0 ||
    strcasecmp(argv[i], fullname2) == 0) {
02152
02153
02154
            sprintf(rval, "%s", argv[i + 1]);
02155
            contain = 1;
02156
            break;
02157
02158
02159
        /* Close file... */
02160
        if (in != NULL)
02161
         fclose(in);
02162
02163
        /* Check for missing variables... */
02164
        if (!contain) {
02165
        if (strlen(defvalue) > 0)
02166
            sprintf(rval, "%s", defvalue);
02167
          else (
            sprintf(msg, "Missing variable %s!\n", fullname1);
02168
02169
            ERRMSG (msq);
02170
          }
02171
02172
        /* Write info... */
printf("%s = %s\n", fullname1, rval);
02173
02174
02175
02176
        /* Return values... */
        if (value != NULL)
02177
02178
         sprintf(value, "%s", rval);
02179
        return atof(rval);
02180 }
02181
```

```
02183
02184 void time2jsec(
02185
       int year,
02186
       int mon,
02187
       int day,
02188
       int hour,
02189
       int min,
02190
       int sec,
02191
       double remain,
02192
       double *jsec) {
02193
02194
       struct tm t0, t1;
02195
02196
       t0.tm_year = 100;
02197
       t0.tm\_mon = 0;
       t0.tm_mday = 1;
t0.tm_hour = 0;
02198
02199
       t0.tm_min = 0;
02200
       t0.tm\_sec = 0;
02201
02202
02203
       t1.tm_year = year - 1900;
       t1.tm_{mon} = mon - 1;
02204
02205
       t1.tm_mday = day;
       t1.tm_hour = hour;
02206
02207
       t1.tm_min = min;
02208
       t1.tm_sec = sec;
02209
02210
       *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
02211 }
02212
02214
02215 void timer(
02216
       const char *name,
02217
       int id,
02218
       int mode) {
02219
       static double starttime[NTIMER], runtime[NTIMER];
02221
       /* Check id... */
if (id < 0 || id >= NTIMER)
02222
02223
        ERRMSG("Too many timers!");
02224
02225
02226
       /* Start timer... */
02227
       if (mode == 1) {
02228
        if (starttime[id] <= 0)</pre>
02229
          starttime[id] = omp_get_wtime();
02230
         else
           ERRMSG("Timer already started!");
02231
02232
02233
       /* Stop timer... */
else if (mode == 2) {
02234
02235
02236
       if (starttime[id] > 0) {
          runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
starttime[id] = -1;
02237
02238
02239
02240
02241
02242
       /* Print timer... */
       else if (mode == 3) {
   printf("%s = %.3f s\n", name, runtime[id]);
02243
02244
02245
         runtime[id] = 0;
02246
02247 }
02248
02250
02251 void write_atm(
02252 const char *filename,
       ctl_t * ctl,
atm_t * atm,
02253
02254
02255
       double t) {
02256
02257
       FILE *in, *out;
02258
02259
       char line[LEN];
02260
02261
       double r, t0, t1;
02262
02263
       int ip, iq, year, mon, day, hour, min, sec;
02264
02265
        /* Set time interval for output... */
02266
       t0 = t - 0.5 * ctl->dt_mod;
02267
       t1 = t + 0.5 * ctl->dt_mod;
02268
02269
       /* Write info... */
```

```
printf("Write atmospheric data: %s\n", filename);
02271
        /* Write ASCII data... */
02272
02273
        if (ctl->atm_type == 0) {
02274
          /* Check if gnuplot output is requested... */
if (ctl->atm_gpfile[0] != '-') {
02275
02276
02277
            /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
02278
02279
               ERRMSG("Cannot create pipe to gnuplot!");
02280
02281
            02282
02283
02284
02285
            /* Set time string... */
            jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02286
02287
                     year, mon, day, hour, min);
02288
02289
02290
             /* Dump gnuplot file to pipe... */
            if (!(in = fopen(ctl->atm_gpfile, "r")))
02291
              ERRMSG("Cannot open file!");
02292
02293
            while (fgets(line, LEN, in))
  fprintf(out, "%s", line);
02294
02295
            fclose(in);
02296
02297
02298
          else {
02299
02300
             /* Create file... */
02301
            if
                (!(out = fopen(filename, "w")))
02302
               ERRMSG("Cannot create file!");
02303
02304
          /* Write header... */
02305
02306
          fprintf(out,
                   "# $1 = time [s] \n"
02307
02308
                   "# $2 = altitude [km] \n"
02309
                   "# $3 = longitude [deg] \n" "# <math>$4 = latitude [deg] \n");
          02310
02311
02312
          fprintf(out, "\n");
02313
02314
02315
          /* Write data... */
02316
          for (ip = 0; ip < atm->np; ip++) {
02317
02318
             /* Check time... */
02319
            if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
02320
              continue;
02321
02322
            /\star Write output... \star/
            fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
    atm->lon[ip], atm->lat[ip]);
02323
02324
            for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02325
02326
02327
               fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02328
02329
            fprintf(out, "\n");
02330
          }
02331
02332
           /* Close file... */
02333
          fclose(out);
02334
02335
02336
        /* Write binary data... */
        else if (ctl->atm_type == 1) {
02337
02338
02339
          /* Create file... */
02340
          if (!(out = fopen(filename, "w")))
            ERRMSG("Cannot create file!");
02341
02342
           /* Write data... */
02343
02344
          FWRITE(&atm->np, int,
02345
02346
                  out);
02347
          FWRITE(atm->time, double,
02348
                   (size_t) atm->np,
02349
                  out):
02350
          FWRITE(atm->p, double,
02351
                   (size_t) atm->np,
02352
                  out);
02353
          FWRITE(atm->lon, double,
02354
                   (size_t) atm->np,
02355
                  out);
          FWRITE(atm->lat, double,
02356
```

```
02357
                     (size_t) atm->np,
02358
                  out);
          for (iq = 0; iq < ctl->nq; iq++)
02359
02360
           FWRITE(atm->q[iq], double,
02361
                      (size_t) atm->np,
02362
                    out);
02363
02364
           /* Close file... */
02365
          fclose(out);
02366
02367
02368
        /* Error... */
02369
        else
02370
          ERRMSG("Atmospheric data type not supported!");
02371 }
02372
02374
02375 void write_csi(
02376
        const char *filename,
02377
        ctl_t * ctl,
        atm t * atm,
02378
02379
        double t) {
02380
02381
        static FILE *in, *out;
02382
02383
        static char line[LEN];
02384
02385
        static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
02386
          rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
02387
02388
        static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
02389
02390
        /* Init... */
02391
        if (t == ctl->t_start) {
02392
          /* Check quantity index for mass... */ if (ctl->qnt_m < 0)
02393
02394
02395
             ERRMSG("Need quantity mass!");
02396
          /* Open observation data file... */
printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
if (!(in = fopen(ctl->csi_obsfile, "r")))
02397
02398
02399
            ERRMSG("Cannot open file!");
02400
02401
02402
           /* Create new file... */
          printf("Write CSI data: %s\n", filename);
if (!(out = fopen(filename, "w")))
02403
02404
            ERRMSG("Cannot create file!");
02405
02406
02407
           /* Write header... */
02408
           fprintf(out,
02409
                   "# $1 = time [s] \n"
                   "# $2 = number of hits (cx)\n"
"# $3 = number of misses (cy)\n"
02410
02411
                   "# $4 = number of false alarms (cz)\n"
02412
02413
                   "# $5 = number of observations (cx + cy) n"
02414
                   "# $6 = number of forecasts (cx + cz)\n"
02415
                   "# \$7 = bias (forecasts/observations) [\%] \n"
                   "# $8 = probability of detection (POD) [%%]\n" # $9 = false alarm rate (FAR) [%%]\n"
02416
02417
                   "# $10 = critical success index (CSI) [%%]\n\n");
02418
02419
02420
02421
        /* Set time interval... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
02422
02423
02424
02425
        /* Initialize grid cells... */
        for (ix = 0; ix < ctl->csi_nx; ix++)
02426
02427
         for (iy = 0; iy < ctl->csi_ny; iy++)
02428
             for (iz = 0; iz < ctl->csi_nz; iz++)
02429
               modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
02430
02431
        /* Read observation data... */
02432
        while (fgets(line, LEN, in)) {
02433
          /* Read data... */
if (sscanf(line, "%lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02434
02435
02436
              5)
02437
             continue;
02438
02439
           /* Check time... */
02440
          if (rt < t0)
          continue;
if (rt > t1)
02441
02442
02443
            break:
```

```
02444
           /* Calculate indices... */
02445
02446
           ix = (int) ((rlon - ctl->csi_lon0))
                        / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02447
02448
           iy = (int) ((rlat - ctl->csi_lat0))
                         / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02449
           iz = (int) ((rz - ctl -> csi_z0))
02450
02451
                        / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02452
           /* Check indices... */
02453
           if (ix < 0 || ix >= ctl->csi_nx ||
02454
               iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02455
02456
             continue;
02457
02458
           /\star Get mean observation index... \star/
02459
           obsmean[ix][iy][iz] += robs;
02460
          obscount[ix][iy][iz]++;
02461
02462
02463
         /* Analyze model data... */
02464
        for (ip = 0; ip < atm->np; ip++) {
02465
02466
           /* Check time... */
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
02467
02468
            continue;
02469
02470
           /* Get indices... */
02471
          ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
02472
                         / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
           iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
02473
02474
                         / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
          02476
02477
          /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
02478
02479
               iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02480
             continue:
02482
02483
           /* Get total mass in grid cell... */
02484
          modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02485
02486
02487
         /* Analyze all grid cells... */
        for (ix = 0; ix < ctl->csi_nx; ix++)
02488
02489
          for (iy = 0; iy < ctl->csi_ny; iy++)
02490
             for (iz = 0; iz < ctl->csi_nz; iz++) {
02491
02492
                /* Calculate mean observation index... */
               if (obscount[ix][iy][iz] > 0)
02493
02494
                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
02495
02496
                /\star Calculate column density... \star/
               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;
02497
02498
02499
                  lat = ctl -> csi_lat0 + dlat * (iy + 0.5);
02500
                 area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180. * cos(lat * M_PI / 180.);
02501
02502
                  modmean[ix][iy][iz] /= (1e6 * area);
02503
02504
               }
02505
02506
               /* Calculate CSI... */
02507
               if (obscount[ix][iy][iz] > 0) {
02508
                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02509
                      modmean[ix][iy][iz] >= ctl->csi_modmin)
02510
                    cx++;
02511
                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
                            modmean[ix][iy][iz] < ctl->csi_modmin)
02512
02514
                  else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
02515
                            modmean[ix][iy][iz] >= ctl->csi_modmin)
02516
                   cz++;
02517
               }
02518
             }
02519
02520
        /* Write output... */
02521
        if (fmod(t, ctl->csi_dt_out) == 0) {
02522
02523
           /* Write... */
          fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
02524
                   (OUT, "$.21 60 80 80 80 80 89 89 89 89 ...,

t, cx, cy, cz, cx + cy, cx + cz,

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

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

(cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
02526
02527
02528
                    (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
02529
02530
```

```
/* Set counters to zero... */
02532
         cx = cy = cz = 0;
02533
02534
02535
        /* Close file... */
if (t == ctl->t_stop)
02536
02537
         fclose(out);
02538 }
02539
02541
02542 void write ens(
02543
        const char *filename,
02544
        ctl_t * ctl,
02545
        atm_t * atm,
02546
        double t) {
02547
02548
        static FILE *out;
       static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
t0, t1, x[NENS][3], xm[3];
02550
02551
02552
02553
        static int ip, iq;
02554
02555
        static size_t i, n;
02556
02557
        /* Init... */
02558
        if (t == ctl->t_start) {
02559
02560
          /* Check quantities... */
          if (ctl->qnt_ens < 0)
02561
02562
            ERRMSG("Missing ensemble IDs!");
02563
02564
          /\star Create new file... \star/
          printf("Write ensemble data: %s\n", filename);
if (!(out = fopen(filename, "w")))
02565
02566
            ERRMSG("Cannot create file!");
02567
02568
02569
          /* Write header... */
02570
          fprintf(out,
                   "# $1 = time [s]\n"
"# $2 = altitude [km]\n"
02571
02572
                   "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02573
          02574
02575
02576
          02577
02578
02579
02580
02581
02582
02583
        /* Set time interval... */
       t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
02584
02585
02586
02587
        /* Init... */
02588
        ens = GSL_NAN;
02589
        n = 0;
02590
02591
        /* Loop over air parcels... */
02592
        for (ip = 0; ip < atm->np; ip++) {
02594
          /* Check time... */
02595
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
02596
            continue;
02597
02598
          /* Check ensemble id... */
02599
          if (atm->q[ctl->qnt_ens][ip] != ens) {
02601
             /* Write results... */
02602
            if (n > 0) {
02603
              /* Get mean position... */
02604
02605
              xm[0] = xm[1] = xm[2] = 0;
for (i = 0; i < n; i++) {
02606
                xm[0] += x[i][0] / (double) n;
xm[1] += x[i][1] / (double) n;
xm[2] += x[i][2] / (double) n;
02607
02608
02609
              }
02610
              cart2geo(xm, &dummy, &lon, &lat);
fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
02611
02612
02613
02614
02615
               /\star Get quantity statistics... \star/
              for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02616
02617
```

```
fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02619
               for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02620
02621
02622
                 fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02623
02624
               fprintf(out, " %lu\n", n);
02625
02626
             /* Init new ensemble... */
02627
02628
             ens = atm->q[ctl->qnt_ens][ip];
             n = 0;
02629
02630
           }
02631
02632
           /* Save data...
02633
           p[n] = atm->p[ip];
02634
           geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
           for (iq = 0; iq < ctl->nq; iq++)
q[iq][n] = atm->q[iq][ip];
02635
02636
           if ((++n) >= NENS)
02637
02638
             ERRMSG("Too many data points!");
02639
02640
        /* Write results... */
02641
02642
        if (n > 0) {
02643
02644
           /\star Get mean position... \star/
02645
           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;</pre>
02646
02647
02648
02649
             xm[2] += x[i][2] / (double) n;
02650
02651
           cart2geo(xm, &dummy, &lon, &lat);
           fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
02652
02653
02654
           /* Get quantity statistics... */
          for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02655
02656
02657
             fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02658
           for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02659
02660
02661
02662
02663
           fprintf(out, " lu\n", n);
02664
02665
        /* Close file... */
02666
        if (t == ctl->t_stop)
02667
02668
          fclose(out);
02669 }
02670
02672
02673 void write grid(
02674
       const char *filename,
02675
        ctl_t * ctl,
02676
        met_t * met0,
02677
        met_t * met1,
        atm t * atm.
02678
02679
        double t) {
02680
02681
        FILE *in, *out;
02682
02683
        char line[LEN];
02684
        static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
area, rho_air, press, temp, cd, vmr, t0, t1, r;
02685
02686
02687
02688
        static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
02689
        /* Check dimensions... */    if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
02690
02691
          ERRMSG("Grid dimensions too large!");
02692
02693
02694
        /* Check quantity index for mass... */
02695
        if (ctl->qnt_m < 0)
          ERRMSG("Need quantity mass!");
02696
02697
02698
        /* Set time interval for output... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
02699
02700
02701
         /* Set grid box size... */
02702
        dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
02703
02704
        dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
```

```
dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
02706
02707
         /* Initialize grid... */
         for (ix = 0; ix < ctl->grid_nx; ix++)
02708
           for (iy = 0; iy < ctl->grid_ny; iy++)
  for (iz = 0; iz < ctl->grid_nz; iz++)
02709
02710
02711
                mass[ix][iy][iz] = 0;
02712
02713
         /* Average data... */
         for (ip = 0; ip < atm->np; ip++)
  if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {</pre>
02714
02715
02716
02717
              /* Get index... */
             ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
02718
02719
02720
              iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
02721
02722
              /* Check indices... */
              if (ix < 0 || ix >= ctl->grid_nx ||
                  iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
02724
02725
02726
             /* Add mass... */
mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02727
02728
02729
02730
02731
         /\star Check if gnuplot output is requested... \star/
02732
         if (ctl->grid_gpfile[0] != '-') {
02733
           /* Write info... */
02734
02735
           printf("Plot grid data: %s.png\n", filename);
02737
            /* Create gnuplot pipe... */
           if (!(out = popen("gnuplot", "w")))
02738
             ERRMSG("Cannot create pipe to gnuplot!");
02739
02740
           /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
02741
02742
02743
02744
            /* Set time string... */
           jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02745
02746
02747
                     year, mon, day, hour, min);
02748
02749
           /* Dump gnuplot file to pipe... */
02750
           if (!(in = fopen(ctl->grid_gpfile, "r")))
             ERRMSG("Cannot open file!");
02751
           while (fgets(line, LEN, in))
fprintf(out, "%s", line);
02752
02753
02754
           fclose(in);
02755
02756
02757
         else {
02758
02759
           /* Write info... */
02760
          printf("Write grid data: %s\n", filename);
02761
02762
           /* Create file... */
02763
           if (!(out = fopen(filename, "w")))
             ERRMSG("Cannot create file!");
02764
02765
02766
02767
         /* Write header... */
02768
        fprintf(out,
02769
                   "# $1 = time [s] \n"
                  "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n"
02770
02771
02772
                   "# $4 = latitude [deg] \n"
02773
                   "# $5 = surface area [km^2]\n"
02774
                   "# $6 = layer width [km] \n"
02775
                   "# $7 = temperature [K] \n"
                   "# $8 = column density [kg/m^2] n"
02776
                   "# $9 = volume mixing ratio [1]\n\n");
02777
02778
02779
         /* Write data... */
02780
         for (ix = 0; ix < ctl->grid_nx; ix++) {
02781
              (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
02782
             fprintf(out, "\n");
           for (iy = 0; iy < ctl->grid_ny; iy++) {
  if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
02783
02784
              fprintf(out, "\n");
for (iz = 0; iz < ctl->grid_nz; iz++)
02785
02787
                if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
02788
                   /* Set coordinates... */
02789
                  z = ctl->grid_z0 + dz * (iz + 0.5);
lon = ctl->grid_lon0 + dlon * (ix + 0.5);
02790
02791
```

```
lat = ctl->grid_lat0 + dlat * (iy + 0.5);
02793
02794
                  /* Get pressure and temperature... */
                  press = P(z);
02795
                  02796
02797
02798
02799
                  /\star Calculate surface area... \star/
                 area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
 * cos(lat * M_PI / 180.);
02800
02801
02802
                 /* Calculate column density... */
cd = mass[ix][iy][iz] / (1e6 * area);
02803
02804
02805
02806
                  /\star Calculate volume mixing ratio... \star/
                  rho_air = 100. * press / (RA * temp);
vmr = MA / ctl->molmass * mass[ix][iy][iz]
02807
02808
                    / (rho_air * 1e6 * area * 1e3 * dz);
02809
02810
                  /* Write output... */
02811
                  fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g,",
t, z, lon, lat, area, dz, temp, cd, vmr);
02812
02813
02814
                }
02815
           }
02816
02817
02818
         /* Close file... */
02819
        fclose(out);
02820 }
02821
02823
02824 void write_prof(
02825
        const char *filename,
        ctl_t * ctl,
met_t * met0,
02826
02827
        met_t * met1,
02828
        atm_t * atm,
02829
02830
        double t) {
02831
02832
        static FILE *in, *out;
02833
02834
        static char line[LEN]:
02835
        static double mass[GX][GY][GZ], obsmean[GX][GY], rt, rz, rlon, rlat, robs,
t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, vmr, h2o,
02836
02837
02838
02839
02840
        static int obscount[GX][GY], ip, ix, iy, iz, okay;
02841
02842
         /* Init... */
02843
         if (t == ctl->t_start) {
02844
          /* Check quantity index for mass... */ if (ctl->qnt_m < 0)
02845
02846
             ERRMSG("Need quantity mass!");
02847
02848
02849
           /* Check dimensions... */
           if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
    ERRMSG("Grid dimensions too large!");
02850
02851
02852
02853
           /* Open observation data file... */
02854
           printf("Read profile observation data: %s\n", ctl->prof_obsfile);
02855
              (!(in = fopen(ctl->prof_obsfile, "r")))
             ERRMSG("Cannot open file!");
02856
02857
           /* Create new output file... */
printf("Write profile data: %s\n", filename);
if (!(out = fopen(filename, "w")))
02858
02859
02860
             ERRMSG("Cannot create file!");
02861
02862
02863
           /* Write header... */
02864
           fprintf(out,
                            = time [s]\n"
                     "# $1
02865
                    "# $2
                           = altitude [km] n"
02866
02867
                           = longitude [deg]\n"
02868
                    "# $4
                           = latitude [deg]\n"
02869
                    "# $5 = pressure [hPa]\n"
                     "# $6 = temperature [K]\n"
02870
02871
                    "# $7 = volume mixing ratio [1]\n"
                    "# $8 = H2O volume mixing ratio [1]\n"
02872
                    "# $9 = 03 volume mixing ratio [1]\n"
02873
02874
                    "# $10 = mean BT index [K]\n");
02875
           /\star Set grid box size... \star/
02876
           dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
02877
02878
```

```
dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
02880
02881
02882
         /\star Set time interval... \star/
         t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
02883
02884
02886
         /* Initialize... */
02887
         for (ix = 0; ix < ctl->prof_nx; ix++)
           for (iy = 0; iy < ctl->prof_ny; iy++) {
02888
              obsmean[ix][iy] = 0;
02889
              obscount[ix][iy] = 0;
for (iz = 0; iz < ctl->prof_nz; iz++)
02890
02891
02892
                mass[ix][iy][iz] = 0;
02893
02894
02895
         /* Read observation data... */
02896
         while (fgets(line, LEN, in)) {
02897
           /* Read data... */
if (sscanf(line, "%lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02898
02899
02900
                5)
02901
              continue;
02902
            /* Check time... */
02903
           if (rt < t0)</pre>
02904
02905
              continue;
           if (rt > t1)
02906
02907
             break;
02908
02909
           /* Calculate indices... */
           ix = (int) ((rlon - ctl->prof_lon0) / dlon);
iy = (int) ((rlat - ctl->prof_lat0) / dlat);
02910
02911
02912
           /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
02913
02914
02915
             continue;
02916
02917
            /* Get mean observation index... */
02918
           obsmean[ix][iy] += robs;
02919
           obscount[ix][iy]++;
         1
02920
02921
02922
         /* Analyze model data... */
02923
         for (ip = 0; ip < atm->np; ip++) {
02924
02925
            /* Check time... */
02926
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
02927
             continue:
02928
02929
            /* Get indices... */
           ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
02930
02931
02932
           iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
02933
02934
            /* Check indices... */
           if (ix < 0 || ix >= ctl->prof_nx ||
02936
                iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
02937
02938
           /* Get total mass in grid cell... */
mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02939
02940
02941
02942
02943
         /* Extract profiles... */
02944
         for (ix = 0; ix < ctl->prof_nx; ix++)
           for (iy = 0; iy < ctl->prof_ny; iy++)
  if (obscount[ix][iy] > 0) {
02945
02946
02947
                /* Check profile... */
                okay = 0;
for (iz = 0; iz < ctl->prof_nz; iz++)
02949
02950
02951
                 if (mass[ix][iy][iz] > 0)
02952
                     okay = 1;
02953
                if (!okav)
02954
                  continue;
02955
                /* Write output... */ fprintf(out, "\n");
02956
02957
02958
02959
                /* Loop over altitudes... */
02960
                for (iz = 0; iz < ctl->prof_nz; iz++) {
02961
02962
                   /* 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);
02963
02964
02965
```

```
02967
                /\star Get pressure and temperature... \star/
02968
                press = P(z);
                02969
02970
02971
02972
                /* Calculate surface area... */
                area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
 * cos(lat * M_PI / 180.);
02973
02974
02975
02976
                /* Calculate volume mixing ratio... */
                rho_air = 100. * press / (RA * temp);
vmr = MA / ctl->molmass * mass[ix][iy][iz]
02977
02978
02979
                  / (rho_air * area * dz * 1e9);
02980
                02981
02982
02983
02984
02985
02986
02987
       /* Close file... */
if (t == ctl->t_stop)
02988
02989
02990
          fclose(out);
02991 }
02992
02994
02995 void write station(
02996
       const char *filename,
       ctl_t * ctl,
atm_t * atm,
02997
02998
02999
       double t) {
03000
       static FILE *out:
03001
03002
       static double rmax2, t0, t1, x0[3], x1[3];
03004
03005
       static int ip, iq;
03006
       /* Init... */
if (t == ctl->t start) {
03007
03008
03009
03010
          /* Write info... */
03011
          printf("Write station data: %s\n", filename);
03012
          /* Create new file... */
if (!(out = fopen(filename, "w")))
03013
03014
            ERRMSG("Cannot create file!");
03015
03016
03017
          /* Write header... */
03018
          fprintf(out,
03019
                  "# $1 = time [s] \n"
                   "# $2 = altitude [km]\n"
03020
                  "# $3 = longitude [deg] \n" "# $4 = latitude [deg] \n");
03021
          for (iq = 0; iq < ctl->nq; iq++)
fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
03022
03023
          ctl->qnt_name[iq], ctl->qnt_unit[iq]);
fprintf(out, "\n");
03024
03025
03026
03027
          /\star Set geolocation and search radius... \star/
03028
          geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
03029
          rmax2 = gsl_pow_2(ctl->stat_r);
03030
03031
       /* Set time interval for output... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
03032
03033
03034
03036
        /* Loop over air parcels... */
03037
        for (ip = 0; ip < atm->np; ip++) {
03038
03039
          /* Check time... */
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
03040
03041
            continue;
03042
03043
          /\star Check station flag... \star/
03044
          if (ctl->qnt_stat >= 0)
03045
           if (atm->q[ctl->qnt_stat][ip])
03046
              continue;
03047
03048
          /* Get Cartesian coordinates... */
03049
          geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
03050
03051
          /* Check horizontal distance... */
          if (DIST2(x0, x1) > rmax2)
03052
```

```
continue;
03055
             /* Set station flag... */
03056
            if (ctl->qnt_stat >= 0)
03057
               atm->q[ctl->qnt_stat][ip] = 1;
03058
            /* Write data... */
fprintf(out, "%.2f %g %g %g",
03059
03060
            atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
for (iq = 0; iq < ctl->nq; iq++) {
   fprintf(out, " ");
   fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
03061
03062
03063
03064
03065
03066
             fprintf(out, "\n");
03067
03068
          /* Close file... */
03069
03070
         if (t == ctl->t_stop)
03071
            fclose(out);
03072 }
```

## 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.

• double deg2dx (double dlon, double lat)

Convert degrees to horizontal distance.

double deg2dy (double dlat)

Convert degrees to horizontal distance.

• double dp2dz (double dp, double p)

Convert pressure to vertical distance.

double dx2deg (double dx, double lat)

Convert horizontal distance to degrees.

• double dy2deg (double dy)

Convert horizontal distance to degrees.

double dz2dp (double dz, double p)

Convert vertical distance to pressure.

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 \*b20, 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 (double \*xx, int n, double x)

Find array index.

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.

- $\bullet \ \ void \ read\_met\_help \ (int \ ncid, \ char \ *varname, \ char \ *varname2, \ \underline{met\_t} \ *met, \ float \ dest[EX][EY][EP], \ float \ scl)$ 
  - 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) 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,
00057
         5, 15, 25, 35, 45, 55, 65, 75, 85
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] = {
             {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00065
               {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},
00066
00067
00068
                {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
                {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},
00070
00071
                {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}, {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
00072
00073
               {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},
00074
00075
00076
00077
                \{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}, {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00078
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},
00080
                {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
00082
              {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03,
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},
00084
00085
00086
00087
                {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
00088
00089
                {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
nnnan
                {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
00091
                \{0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167\},
00092
               {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},
00093
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
00096
                {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}, {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
00097
00098
00099
                {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
00101
              {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
               \{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}, {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},
00103
00104
00105
                {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},
               \{0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135\}
00111
               \{0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194\},\
00112
               \{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}, 
{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
00114
00115
00116
00117
00118
               {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00120
                {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
00121
               {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},
00124
               {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}
00126
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}, {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
00130
               \{0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02\},
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},
00133
00134
                {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},
              {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},
00139
00140
                {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},
00141
00142
               {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
00143
                {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
                {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}, {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
              \{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},
              {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
00160
              {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217}
00161
              {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
00162
              {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
00163
00164
              {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},
00165
00166
              (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},
00167
              {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
              {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}},
00170
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
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},
00176
              {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
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}, {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
00178
00179
00180
00181
              {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254,
              {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
00182
00183
              {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25}
00184
              \{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},
00185
              {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913}
00186
              {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
00188
              \{0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56\},
              {0.735, 1.51, 2.96, 4.84, 5.92,
                                                       7.77,
                                                               7.2, 5.54, 2.56, 1.61},
00189
             {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}
00195
              {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
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},
00199
              {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},
00201
00202
              {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},
              {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754}, {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
00204
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
              {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
00208
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},
00210
00211
              \{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},
00213
00214
00215
              {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
00216
              {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},
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},
              {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
00220
00221
              {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}, {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},
00222
00223
00224
00225
               {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
00226
00227
             {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
00228
              {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
              \{0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97\},
00229
              {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
              {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},
00234
              {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},
00235
00236
```

```
00237
               {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}, {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
00238
00239
00240
               \{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},
00241
00242
               {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},
                                                                                    2.42, 1.73},
00246
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\},
               {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
00250
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}, {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}, {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},
00256
00257
00258
00259
               {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},
00260
00261
               {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},
00263
               {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
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
00268
               \{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}, {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
00269
00270
               {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},
00271
00272
00273
               {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
00275
               \{0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6\},\
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
00281
00282
00283
          double aux00, aux01, aux10, aux11, sec;
00284
          int ilat, ip, isec;
00285
00286
           /* Get seconds since begin of year... */
          sec = fmod(t, 365.25 * 86400.);
00288
00289
00290
           /* Get indices... */
          ilat = locate(lats, 18, lat);
00291
          ip = locate(ps, 10, p);
00292
           isec = locate(secs, 12, sec);
00293
00294
00295
           /* Interpolate...
00296
          aux00 = LIN(ps[ip], hno3[isec][ilat][ip],
                           ps[ip + 1], hno3[isec][ilat][ip + 1], p);
00297
          00298
00299
           aux10 = LIN(ps[ip], hno3[isec + 1][ilat][ip],
00300
00301
                           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);
          aux00 = LIN(lats[ilat], aux00, lats[ilat + 1], aux01, lat);
aux11 = LIN(lats[ilat], aux10, lats[ilat + 1], aux11, lat);
return LIN(secs[isec], aux00, secs[isec + 1], aux11, sec);
00304
00305
00307 }
```

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
00314
00315
          static double doys[12]
00316
          = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00317
00318
          static double lats[73]
               \{-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,
00320
00321
            -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5
            -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
            75, 77.5, 80, 82.5, 85, 87.5, 90
00325
00326
00327
00328
          static double tps[12][73]
            = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6, 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,
00329
00330
                   99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
00332
                   98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5,
00333
00334
                   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,
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,
00336
00337
           300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
           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,
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
00345
00346
           297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
           161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3, 100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
00347
00348
           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
00351
           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},
00352
00353
          {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, 102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
00354
00355
00356
           99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
00358
           148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
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,
                                                                                 104.1,
           101.8, 101.4, 101.1, 101, 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}, {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
00368
           222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
00370
           228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
00371
00372
           105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
           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
          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,
00376
00377
00378
           187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
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
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
           112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
00389
00390
           120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
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,
00393
           183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
           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},
            {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
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
                                                                                           111.2,
00405
             112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
00407
             206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
00408
             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, 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,
00410
00411
00412
             108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
00413
00414
             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, 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,
00415
00416
00417
             284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
00419
             261.7, 261.3, 271.1, 270.4, 261.7, 230.6, 231.6, 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, 99.31, 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, 280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
00420
00421
00422
00423
00424
             281.7, 281.1, 281.2}
00425
00426
00427
00428
           double doy, p0, p1, pt;
00429
00430
           int imon, ilat;
00431
00432
            /* Get day of year... */
00433
           doy = fmod(t / 86400., 365.25);
           while (doy < 0)</pre>
00434
              doy += 365.25;
00435
00436
00437
            /* Get indices... */
00438
            imon = locate(doys, 12, doy);
00439
           ilat = locate(lats, 73, lat);
00440
00441
            /* Get tropopause pressure... */
           00442
00443
           p1 = LIN(lats[ilat], tps[imon + 1][ilat],
lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
00445
00446
           pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
00447
00448
           /* Return tropopause pressure... */
00449
           return pt;
00450 }
```

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 454 of file libtrac.c.

```
00458
00459
          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 \};
00460
00461
00462
00463
          if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
00464
00465
            *doy = d01[mon - 1] + day - 1;
00466
          else
00467
             *dov = d0[mon - 1] + dav - 1;
00468 }
```

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

Get date from day of year.

Definition at line 472 of file libtrac.c.

```
00476
00477
            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 \};
00478
00479
00481
            /* 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>
00482
00483
00484
00485
            break;

*mon = i + 1;

*day = doy - d01[i] + 1;
00487
00488
00489 } else {
            for (i = 11; i >= 0; i--)
   if (d0[i] <= doy)</pre>
00490
00491
             break;

*mon = i + 1;

*day = doy - d0[i] + 1;
00492
00493
00494
00496 }
```

5.21.2.6 double deg2dx ( double dlon, double lat )

Convert degrees to horizontal distance.

Definition at line 500 of file libtrac.c.

5.21.2.7 double deg2dy ( double dlat )

Convert degrees to horizontal distance.

Definition at line 509 of file libtrac.c.

```
00510 {
00511
00512 return dlat * M_PI * RE / 180.;
00513 }
```

5.21.2.8 double dp2dz ( double dp, double p )

Convert pressure to vertical distance.

Definition at line 517 of file libtrac.c.

```
00519 {
00520
00521 return -dp * H0 / p;
00522 }
```

```
5.21.2.9 double dx2deg ( double dx, double lat )
```

Convert horizontal distance to degrees.

Definition at line 526 of file libtrac.c.

5.21.2.10 double dy2deg ( double dy )

Convert horizontal distance to degrees.

Definition at line 539 of file libtrac.c.

```
00540 {
00541
00542 return dy * 180. / (M_PI * RE);
00543 }
```

5.21.2.11 double dz2dp ( double dz, double p )

Convert vertical distance to pressure.

Definition at line 547 of file libtrac.c.

```
00549 {
00550
00551 return -dz * p / H0;
00552 }
```

5.21.2.12 void geo2cart ( double z, double lon, double lat, double \*x )

Convert geolocation to Cartesian coordinates.

Definition at line 556 of file libtrac.c.

```
00560 {
00561
00562 double radius;
00563
00564 radius = z + RE;
00565 x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
00566 x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
00567 x[2] = radius * sin(lat / 180 * M_PI);
```

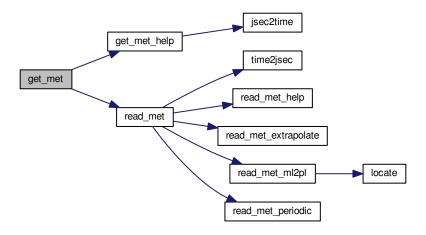
5.21.2.13 void get\_met ( ctl\_t \* ctl, char \* metbase, double t, met\_t \* met0, met\_t \* met1 )

Get meteorological data for given timestep.

Definition at line 572 of file libtrac.c.

```
00577
00578
        static int init;
00580
00581
        char filename[LEN];
00582
00583
        /* Init... */
        if (t == ctl->t_start || !init) {
  init = 1;
00584
00585
00586
00587
          get_met_help(t, -1, metbase, ctl->dt_met, filename);
00588
          read_met(ctl, filename, met0);
00589
          get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
00590
     dt_met, filename);
00591
          read_met(ctl, filename, met1);
00592
00593
        /\star Read new data for forward trajectories... \star/
00594
00595
        if (t > met1->time && ct1->direction == 1) {
        memcpy(met0, met1, sizeof(met_t));
00596
         get_met_help(t, 1, metbase, ctl->dt_met, filename);
read_met(ctl, filename, metl);
00598
00599
00600
        /* Read new data for backward trajectories... */
00601
        if (t < met0->time && ctl->direction == -1) {
00602
         memcpy(met1, met0, sizeof(met_t));
00604
          get_met_help(t, -1, metbase, ctl->dt_met, filename);
00605
          read_met(ctl, filename, met0);
00606
00607 }
```

Here is the call graph for this function:



5.21.2.14 void get\_met\_help ( double t, int direct, char \* metbase, double dt\_met, char \* filename )

Get meteorological data for timestep.

Definition at line 611 of file libtrac.c.

```
{
00618
        double t6, r;
00619
00620
        int year, mon, day, hour, min, sec;
00621
00622
        /\star Round time to fixed intervals... \star/
00623
        if (direct == -1)
00624
          t6 = floor(t / dt_met) * dt_met;
00625
        else
00626
          t6 = ceil(t / dt_met) * dt_met;
00627
00628
        /* Decode time... */
00629
        jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00630
        /* Set filename... */
sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", metbase, year, mon, day, hour);
00631
00632
00633 }
```

Here is the call graph for this function:



5.21.2.15 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 637 of file libtrac.c.

```
00643
00644
00645
           double aux00, aux01, aux10, aux11;
00646
00647
           /* Set variables...
          /* Set variables...*/
aux00 = array[ix][iy];
aux01 = array[ix][iy + 1];
aux10 = array[ix + 1][iy];
aux11 = array[ix + 1][iy + 1];
00648
00650
00651
00652
00653
           /* Interpolate horizontally... */
           aux00 = wy * (aux00 - aux01) + aux01;
aux11 = wy * (aux10 - aux11) + aux11;
00654
00655
00656
            *var = wx * (aux00 - aux11) + aux11;
00657 }
```

5.21.2.16 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 661 of file libtrac.c.

```
00670
00671
         double aux00, aux01, aux10, aux11;
00672
00673
        /* Interpolate vertically... */
        aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00674
          + array[ix][iy][ip + 1];
        aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
00676
        + array[ix][iy + 1][ip + 1];
aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
00677
00678
        + array[ix + 1][iy][ip + 1];
aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
00679
00680
00681
         + array[ix + 1][iy + 1][ip + 1];
00682
00683
        /* Interpolate horizontally... */
00684    aux00 = wy * (aux00 - aux01) + aux01;
00685    aux11 = wy * (aux10 - aux11) + aux11;
00686
        *var = wx * (aux00 - aux11) + aux11;
00687 }
```

5.21.2.17 void intpol\_met\_space ( met\_t \* met, double p, double lon, double lat, double \* ps, double \* pt, double \* z, double \* t, double \* u, double \* v, double \* w, double \* pv, double \* h2o, double \* o3 )

Spatial interpolation of meteorological data.

Definition at line 691 of file libtrac.c.

```
00705
00706
00707
        double wp, wx, wy;
00708
00709
        int ip, ix, iy;
00710
       /* Check longitude... */
if (met->lon[met->nx - 1] > 180 && lon < 0)
00711
00712
00713
          lon += 360;
00714
00715
        /* Get indices... */
00716
       ip = locate(met->p, met->np, p);
        ix = locate(met->lon, met->nx, lon);
00717
00718
        iy = locate(met->lat, met->ny, lat);
00719
00720
00721
       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]);
00722
00723
00724
00725
        /* Interpolate... */
00726
       if (ps != NULL)
00727
          intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
00728
        if (pt != NULL)
00729
          intpol_met_2d(met->pt, ix, iy, wx, wy, pt);
00730
        if (z != NULL)
          intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy, z);
00732
        if (t != NULL)
00733
          intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
00734
       if (u != NULL)
00735
          intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
00736
        if (v != NULL)
00737
          intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
00738
        if (w != NULL)
00739
          intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
00740
       if (pv != NULL)
00741
          intpol_met_3d(met->pv, ip, ix, iy, wp, wx, wy, pv);
00742
        if (h2o != NULL)
00743
          intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
00744
        if (o3 != NULL)
00745
          intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00746 }
```

Here is the call graph for this function:

5.21.2.18 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 750 of file libtrac.c.

```
00766
00768
        double h2o0, h2o1, o30, o31, ps0, ps1, pt0, pt1, pv0, pv1, t0, t1, u0, u1,
00769
          v0, v1, w0, w1, wt, z0, z1;
00770
00771
        /* \ {\tt Spatial interpolation...}
        00772
00773
00774
                          pt == NULL ? NULL : &pt0,
00775
                          z == NULL ? NULL : &z0,
00776
                          t == NULL ? NULL : &t0,
00777
                          u == NULL ? NULL : &u0,
00778
                          v == NULL ? NULL : &v0,
00779
                          w == NULL ? NULL : &w0,
00780
                          pv == NULL ? NULL : &pv0,
00781
                          h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00782
        intpol_met_space(met1, p, lon, lat,
00783
                          ps == NULL ? NULL : &ps1,
                          pt == NULL ? NULL : &pt1,
00784
00785
                          z == NULL ? NULL : &z1,
00786
                          t == NULL ? NULL : &t1,
00787
                          u == NULL ? NULL : &u1,
                          v == NULL ? NULL : &v1,
00788
00789
                          w == NULL ? NULL : &w1,
                         pv == NULL ? NULL : &pv1,
h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00790
00791
00792
00793
        /* Get weighting factor... */
00794
       wt = (met1->time - ts) / (met1->time - met0->time);
00795
00796
        /* Interpolate... */
       if (ps != NULL)
00797
00798
          *ps = wt * (ps0 - ps1) + ps1;
00799
        if (pt != NULL)
       *pt = wt * (pt0 - pt1) + pt1;
if (z != NULL)
00800
00801
00802
         *z = wt * (z0 - z1) + z1;
       if (t != NULL)
00803
00804
         *t = wt * (t0 - t1) + t1;
00805
       if (u != NULL)
00806
          *u = wt * (u0 - u1) + u1;
       if (v != NULL)
00807
         *v = wt * (v0 - v1) + v1;
00808
       if (w != NULL)

*w = wt * (w0 - w1) + w1;
00809
00810
        if (pv != NULL)
00811
00812
          \star pv = wt \star (pv0 - pv1) + pv1;
00813
        if (h2o != NULL)
00814
          *h2o = wt * (h2o0 - h2o1) + h2o1;
        if (o3 != NULL)
00815
          *o3 = wt * (o30 - o31) + o31;
00816
00817 }
```

Here is the call graph for this function:

5.21.2.19 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 821 of file libtrac.c.

```
00829 {
00830
00831 struct tm t0, *t1;
00832
00833 time_t jsec0;
00834
00835 t0.tm_year = 100;
```

```
00836
        t0.tm_mon = 0;
        t0.tm_mday = 1;
t0.tm_hour = 0;
00838
        t0.tm_min = 0;
00839
        t0.tm_sec = 0;
00840
00841
00842
        jsec0 = (time_t) jsec + timegm(&t0);
00843
        t1 = gmtime(&jsec0);
00844
00845
        *year = t1->tm_year + 1900;
        *mon = t1->tm_mon + 1;
*day = t1->tm_mday;
00846
00847
        *hour = t1->tm_hour;
00848
        *min = t1->tm_min;
00849
00850
        *sec = t1->tm_sec;
00851
       *remain = jsec - floor(jsec);
00852 1
```

#### 5.21.2.20 int locate ( double \*xx, int n, double x )

Find array index.

Definition at line 856 of file libtrac.c.

```
00859
                    {
00860
00861
        int i, ilo, ihi;
00863
        ilo = 0;
00864
        ihi = n - 1;
        i = (ihi + ilo) >> 1;
00865
00866
00867
        if (xx[i] < xx[i + 1])
          while (ihi > ilo + 1) {
  i = (ihi + ilo) >> 1;
00870
             if (xx[i] > x)
00871
              ihi = i;
00872
            else
              ilo = i;
00873
00874
        } else
00875
         while (ihi > ilo + 1) {
            i = (ihi + ilo) >> 1;
00876
            if (xx[i] <= x)
  ihi = i;</pre>
00877
00878
00879
            else
00880
              ilo = i;
00881
00882
00883
        return ilo;
00884 }
```

5.21.2.21 void read\_atm ( const char \* filename, ctl\_t \* ctl, atm\_t \* atm )

Read atmospheric data.

Definition at line 888 of file libtrac.c.

```
00891
00892
00893
       FILE *in;
00894
00895
       char line[LEN], *tok;
00896
00897
       double t0:
00898
00899
       int dimid, ip, iq, ncid, varid;
00900
00901
       size_t nparts;
00902
00903
       /* Init... */
00904
       atm->np = 0;
00905
00906
       /* Write info... */
       printf("Read atmospheric data: %s\n", filename);
```

```
00908
00909
          /* Read ASCII data... */
00910
          if (ctl->atm_type == 0) {
00911
00912
            /* Open file... */
if (!(in = fopen(filename, "r")))
00913
00914
               ERRMSG("Cannot open file!");
00915
00916
             /* Read line... */
            while (fgets(line, LEN, in)) {
00917
00918
               /* 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]);
00919
00920
00921
00922
00923
00924
00925
00927
               /* Convert altitude to pressure... */
00928
               atm \rightarrow p[atm \rightarrow np] = P(atm \rightarrow p[atm \rightarrow np]);
00929
               /* Increment data point counter... */
if ((++atm->np) > NP)
00930
00931
00932
                  ERRMSG("Too many data points!");
00933
00934
00935
             /* Close file... */
00936
            fclose(in);
00937
00938
00939
          /* Read binary data... */
00940
          else if (ctl->atm_type == 1) {
00941
00942
             /* Open file... */
            if (!(in = fopen(filename, "r")))
00943
               ERRMSG("Cannot open file!");
00944
00946
             /* Read data... */
00947
            FREAD(&atm->np, int,
00948
                    1.
00949
                    in):
00950
            FREAD (atm->time, double,
00951
                      (size_t) atm->np,
00952
                    in);
00953
            FREAD(atm->p, double,
00954
                      (size_t) atm->np,
00955
                    in);
00956
            FREAD(atm->lon, double,
00957
                      (size_t) atm->np,
00958
                    in);
00959
            FREAD(atm->lat, double,
00960
                      (size_t) atm->np,
            in);
for (iq = 0; iq < ctl->nq; iq++)
FREAD(atm->q[iq], double,
00961
00962
00963
00964
                         (size_t) atm->np,
00965
00966
             /* Close file... */
00967
           fclose(in);
00968
00969
00970
00971
          /* Read netCDF data... */
00972
          else if (ctl->atm_type == 2) {
00973
00974
             /* Open file... */
            NC(nc_open(filename, NC_NOWRITE, &ncid));
00975
00976
00977
             /* Get dimensions... */
00978
            NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
00979
            NC(nc_inq_dimlen(ncid, dimid, &nparts));
            atm->np = (int) nparts;
if (atm->np > NP)
00980
00981
               ERRMSG("Too many particles!");
00982
00983
00984
00985
            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;
00986
00987
00988
            /* Read geolocations... */
NC(nc_inq_varid(ncid, "PRESS", &varid));
00990
00991
            NC(nc_get_var_double(ncid, varid, atm->p));
NC(nc_inq_varid(ncid, "LON", &varid));
00992
00993
00994
            NC(nc_get_var_double(ncid, varid, atm->lon));
```

```
NC(nc_inq_varid(ncid, "LAT", &varid));
00996
           NC(nc_get_var_double(ncid, varid, atm->lat));
00997
00998
            /* Read variables... */
00999
           if (ctl->qnt_p >= 0)
             if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
01000
                NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
01002
           if (ctl->qnt_t >= 0)
01003
             if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
01004
               NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
01005
           if (ctl->qnt_u >= 0)
             if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
01006
           NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
if (ctl->qnt_v >= 0)
01007
01008
01009
             if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
           NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
if (ctl->qnt_w >= 0)
01010
01011
            if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
01012
               NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
01013
           if (ctl->qnt_h2o >= 0)
01014
            if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
01015
01016
01017
           if (ctl->qnt_o3 >= 0)
           if (ccl->qnt_03 >= 0)
  if (nc_inq_varid(ncid, "03", &varid) == NC_NOERR)
   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)
01018
01019
01020
01021
01022
               NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
01023
           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]));
01024
01025
01026
01027
           for (ip = 0; ip < atm\rightarrownp; ip++)
01028
             01029
01030
01031
01032
                   || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
01034
                atm->time[ip] = GSL_NAN;
                atm->p[ip] = GSL_NAN;
atm->lon[ip] = GSL_NAN;
01035
01036
                atm->lat[ip] = GSL_NAN;
for (iq = 0; iq < ctl->nq; iq++)
01037
01038
                 atm->q[iq][ip] = GSL_NAN;
01039
01040
             } else {
01041
               if (ct1->qnt_h2o >= 0)
01042
                  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)
01043
01044
01045
01046
                  atm->lon[ip] -= 360;
01047
01048
            /* Close file... */
01049
01050
           NC(nc close(ncid));
01051
01052
01053
         /* Error... */
01054
01055
           ERRMSG("Atmospheric data type not supported!");
01056
01057
         /* Check number of points... */
01058
         if (atm->np < 1)
01059
           ERRMSG("Can not read any data!");
01060 }
```

5.21.2.22 void read\_ctl ( const char \* filename, int argc, char \* argv[], ctl\_t \* ctl )

Read control parameters.

Definition at line 1064 of file libtrac.c.

```
01068 {
01069
01070 int ip, iq;
01071
01072 /* Write info... */
01073 printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
01074 "(executable: %s | compiled: %s, %s)\n\n",
```

```
argv[0], __DATE__, __TIME__);
01076
01077
        /* Initialize quantity indices... */
01078
        ctl->qnt_ens = -1;
        ctl->qnt_m = -1;
01079
        ctl->qnt_r = -1;
01080
        ctl->qnt_rho = -1;
01082
        ctl->qnt_ps = -1;
01083
        ctl->qnt_pt = -1;
        ctl->qnt_z = -1;
01084
        ctl->qnt_p = -1;
01085
        ctl->qnt_t = -1;
01086
01087
        ctl->qnt_u = -1;
01088
        ctl->qnt_v = -1;
01089
        ctl->qnt_w = -1;
01090
        ctl->qnt_h2o = -1;
        ctl->qnt_o3 = -1;
01091
01092
        ctl->qnt theta = -1;
01093
        ctl->qnt_vh = -1;
01094
        ctl->qnt_vz = -1;
01095
        ctl->qnt_pv = -1;
01096
        ctl->qnt\_tice = -1;
        ctl->qnt\_tsts = -1;
01097
        ctl->qnt_tnat = -1;
01098
01099
        ctl->qnt_stat = -1;
01100
01101
        /* Read quantities... */
01102
        ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
01103
        if (ctl->nq > NQ)
         ERRMSG("Too many quantities!");
01104
01105
        for (iq = 0; iq < ctl->nq; iq++) {
01106
01107
           /* 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",
01108
01109
01110
                    ctl->qnt_format[iq]);
01111
          /* Try to identify quantity... */
if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
01112
01113
01114
            ctl->qnt_ens = iq;
            sprintf(ctl->qnt_unit[iq], "-");
01115
          } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
01116
            ctl->qnt_m = iq;
01117
01118
            sprintf(ctl->qnt_unit[iq], "kg");
          } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
01119
            ctl->qnt_r = iq;
01120
01121
            sprintf(ctl->qnt_unit[iq], "m");
          } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
ctl->qnt_rho = iq;
01122
01123
01124
            sprintf(ctl->qnt_unit[iq], "kg/m^3");
01125
          } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
01126
             ctl->qnt_ps = iq;
01127
            sprintf(ctl->qnt_unit[iq], "hPa");
01128
          } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
            ctl->qnt_pt = iq;
01129
            sprintf(ctl->qnt_unit[iq], "hPa");
01130
          } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
01131
01132
            ctl->qnt_z = iq;
01133
            sprintf(ctl->qnt_unit[iq], "km");
          } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
ctl->qnt_p = iq;
01134
01135
            sprintf(ctl->qnt_unit[iq], "hPa");
01136
          } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
01137
           ctl->qnt_t = iq;
01138
01139
            sprintf(ctl->qnt_unit[iq], "K");
01140
          } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
01141
            ctl->qnt_u = iq;
            sprintf(ctl->qnt_unit[iq], "m/s");
01142
01143
          } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
01144
            ctl->qnt_v = iq;
01145
            sprintf(ctl->qnt_unit[iq], "m/s");
01146
          else\ if\ (strcmp(ctl->qnt_name[iq], "w") == 0) {
            ctl->qnt_w = iq;
sprintf(ctl->qnt_unit[iq], "hPa/s");
01147
01148
          } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
  ctl->qnt_h2o = iq;
01149
01150
01151
            sprintf(ctl->qnt_unit[iq], "1");
01152
          } else if (strcmp(ct1->qnt_name[iq], "o3") == 0) {
01153
            ct1->qnt_o3 = iq;
            sprintf(ctl->qnt_unit[iq], "1");
01154
          } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
01155
            ctl->qnt_theta = iq;
sprintf(ctl->qnt_unit[iq], "K");
01156
01157
01158
          } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
            ctl->qnt_vh = iq;
01159
            sprintf(ctl->qnt_unit[iq], "m/s");
01160
01161
          } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
```

```
01162
             ctl->qnt_vz = iq;
              sprintf(ctl->qnt_unit[iq], "m/s");
01163
01164
            } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
              ctl->qnt_pv = iq;
01165
01166
              sprintf(ctl->qnt_unit[iq], "PVU");
           } else if (strcmp(ct1->qnt_name[iq], "tice") == 0) {
ct1->qnt_tice = iq;
01167
01168
01169
              sprintf(ctl->qnt_unit[iq], "K");
01170
           } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
             ctl->qnt_tsts = iq;
01171
             sprintf(ctl->qnt_unit[iq], "K");
01172
           } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
ctl->qnt_tnat = iq;
01173
01174
01175
              sprintf(ctl->qnt_unit[iq], "K");
           } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
01176
            ctl->qnt_stat = iq;
01177
              sprintf(ctl->qnt_unit[iq], "-");
01178
01179
           } else
01180
             scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
01181
01182
01183
         /\star Time steps of simulation... \star/
01184
         ctl->direction =
           (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
f (ctl->direction != -1 && ctl->direction != 1)
01185
01186
           ERRMSG("Set DIRECTION to -1 or 1!");
01187
         ctl->t_stop = scan_ctl(filename, argo, argv, "T_STOP", -1, "1e100", NULL);
ctl->dt_mod = scan_ctl(filename, argo, argv, "DT_MOD", -1, "600", NULL);
01188
01189
01190
01191
         /* Meteorological data... */
         ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
01192
         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);
01193
01194
         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);
ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
01195
01196
01197
01198
         ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
01199
01200
         if
            (ctl->met_np > EP)
           ERRMSG("Too many levels!");
01201
01202
         for (ip = 0; ip < ctl->met_np; ip++)
           ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
01203
01204
         ctl->met tropo
01205
           = (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);
01206
01207
01208
01209
         /* Isosurface parameters... */
01210
         ctl->isosurf
         = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
01211
01212
01213
01214
         /* Diffusion parameters... */
01215
         ctl->turb_dx_trop
           = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
01216
01217
         ctl->turb dx strat
01218
            = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
01219
         ctl->turb dz trop
01220
            = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
01221
         ctl->turb_dz_strat
01222
           = scan ctl(filename, argc, argv, "TURB DZ STRAT", -1, "0.1", NULL);
01223
         ctl->turb mesox =
01224
           scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
01225
01226
           scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
01227
01228
         /* Mass and life time...
         ctl->molmass = scan_ctl(filename, argc, argv, "MOLMASS", -1, "1", NULL);
01229
         ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
01230
01231
         ctl->tdec strat =
           scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
01232
01233
01234
         /* PSC analysis... */
         ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL); ctl->psc_hno3 =
01235
01236
01237
           scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
01238
         /* Output of atmospheric data... */
scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
01239
01240
      atm basename):
01241
         scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
01242
         ctl->atm_dt_out
01243
           scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
01244
         ctl->atm_filter
01245
           (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
         ctl->atm_type :
01246
01247
            (int) scan ctl(filename, argc, argv, "ATM TYPE", -1, "0", NULL);
```

```
01248
01249
         /* Output of CSI data... */
         scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
01250
      csi basename);
01251 ctl->csi dt out =
         scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->
01252
01253
      csi_obsfile);
01254 ctl->csi_obsmin =
01255
           scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
01256
         ctl->csi modmin =
           scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
01257
         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);
01258
01259
01260
01261
         ctl->csi_lon0 =
         scan_ctl(filename, argc, argv, "CSI_LONO", -1, "-180", NULL);
ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1,
01262
                                                                                       "180", NULL);
01263
01264
         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);
01265
01266
01267
01268
         ctl->csi nv =
           (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
01269
01270
01271
         /* Output of ensemble data... */
01272
         scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
      ens_basename);
01273
01274
         /* Output of grid data... */
         scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
01275
                    ctl->grid_basename);
         scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
01277
      grid_gpfile);
01278
         ctl->grid_dt_out =
           scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
01279
01280
         ctl->grid sparse =
           (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
         ctl-ygrid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL); ctl-ygrid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
01282
01283
         ctl->grid_nz =
01284
           (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
01285
01286
         ct1->grid lon0 =
01287
           scan_ctl(filename, argc, argv, "GRID_LONO", -1, "-180", NULL);
01288
         ctl->grid lon1 =
01289
            scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
         ctl->grid_nx =
01290
           (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
01291
01292
         ctl->grid lat0 =
01293
           scan_ctl(filename, argc, argv, "GRID_LATO", -1, "-90", NULL);
01294
         ctl->grid_lat1
01295
            scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
01296
         ctl->grid_ny =
01297
           (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
01298
01299
         /* Output of profile data... */
         scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
01300
01301
                     ctl->prof_basename);
         scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
01302
prof_obsfile);
01303 ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
01304 ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
01305
         ctl->prof_nz =
01306
            (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
         ctl->prof_lon0 =
01307
01308
           scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
01309
         ctl->prof lon1
           scan ctl(filename, argc, argv, "PROF LON1", -1, "180", NULL);
01310
01311
         ctl->prof_nx =
01312
            (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
01313
          ctl->prof_lat0 =
01314
           scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
01315
         ctl->prof_lat1 =
           scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
01316
01317
         ctl->prof ny :
01318
            (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
01319
01320
         /* Output of station data... */
         scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
01321
01322
                    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);
01323
01324
01325
01326 }
```



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

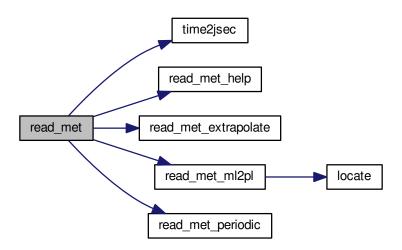
Read meteorological data file.

Definition at line 1330 of file libtrac.c.

```
01333
01334
01335
        char cmd[2 * LEN], levname[LEN], tstr[10];
01336
        static float help[EX * EY];
01337
01338
        int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
01340
01341
        size_t np, nx, ny;
01342
        /* Write info... */
01343
01344
        printf("Read meteorological data: %s\n", filename);
01345
01346
         /\star Get time from filename... \star/
01347
        sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
        year = atoi(tstr);
01348
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
01349
01350
        mon = atoi(tstr);
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
01351
01352
        day = atoi(tstr);
01353
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
01354
        hour = atoi(tstr);
01355
        time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
01356
        /* Open netCDF file... */
01357
01358
        if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
01359
01360
          /\star Try to stage meteo file...
01361
          if (ctl->met_stage[0] != '-') {
   sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
01362
             year, mon, day, hour, filename);
if (system(cmd) != 0)
01363
01364
01365
              ERRMSG("Error while staging meteo data!");
01366
01367
           /* Try to open again... */
01368
          NC(nc_open(filename, NC_NOWRITE, &ncid));
01369
01371
         /* Get dimensions... */
01372
        NC(nc_inq_dimid(ncid, "lon", &dimid));
NC(nc_inq_dimlen(ncid, dimid, &nx));
if (nx < 2 || nx > EX)
01373
01374
01375
01376
          ERRMSG("Number of longitudes out of range!");
01377
01378
        NC(nc_inq_dimid(ncid, "lat", &dimid));
01379
        NC(nc_inq_dimlen(ncid, dimid, &ny));
01380
        if (ny < 2 || ny > EY)
          ERRMSG("Number of latitudes out of range!");
01381
01382
        sprintf(levname, "lev");
01383
01384
        NC(nc_inq_dimid(ncid, levname, &dimid));
01385
        NC(nc_inq_dimlen(ncid, dimid, &np));
01386
        if (np == 1) {
01387
          sprintf(levname, "lev_2");
01388
          NC(nc_inq_dimid(ncid, levname, &dimid));
01389
          NC(nc_inq_dimlen(ncid, dimid, &np));
```

```
01390
01391
         if (np < 2 || np > EP)
01392
           ERRMSG("Number of levels out of range!");
01393
01394
         /* Store dimensions... */
         met->np = (int) np;
met->nx = (int) nx;
01395
01396
01397
         met->ny = (int) ny;
01398
         /* Get horizontal grid... */
NC(nc_inq_varid(ncid, "lon", &varid));
01399
01400
01401
         NC(nc_get_var_double(ncid, varid, met->lon));
NC(nc_inq_varid(ncid, "lat", &varid));
01402
01403
         NC(nc_get_var_double(ncid, varid, met->lat));
01404
        /* 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);
01405
01406
01407
01408
         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));
01409
01410
01411
01412
01413
         /* Meteo data on pressure levels... */
01414
         if (ctl->met_np <= 0) {</pre>
01415
01416
            /* Read pressure levels from file...
01417
           NC(nc_inq_varid(ncid, levname, &varid));
01418
           NC(nc\_get\_var\_double(ncid, varid, met->p));
01419
           for (ip = 0; ip < met->np; ip++)
             met->p[ip] /= 100.;
01420
01421
01422
            /* Extrapolate data for lower boundary... */
01423
           read_met_extrapolate(met);
01424
01425
01426
         /* Meteo data on model levels... */
         else {
01428
           /* Read pressure data from file... */
read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
01429
01430
01431
           /\star Interpolate from model levels to pressure levels... \star/
01432
01433
           read_met_ml2pl(ctl, met, met->t);
           read_met_ml2pl(ctl, met, met->u);
01434
01435
            read_met_ml2pl(ctl, met, met->v);
01436
           read_met_ml2pl(ctl, met, met->w);
01437
           read_met_ml2pl(ctl, met, met->h2o);
01438
           read_met_ml2pl(ctl, met, met->o3);
01439
01440
           /* Set pressure levels... */
           met->np = ctl->met_np;
for (ip = 0; ip < met->np; ip++)
01441
01442
01443
             met->p[ip] = ctl->met_p[ip];
01444
01445
01446
         /* Check ordering of pressure levels... */
         for (ip = 1; ip < met->np; ip++)
  if (met->p[ip - 1] < met->p[ip])
01447
01448
01449
              ERRMSG("Pressure levels must be descending!");
01450
01451
         01452
01453
01454
           NC(nc_get_var_float(ncid, varid, help));
01455
            for (iy = 0; iy < met->ny; iy++)
              for (ix = 0; ix < met \rightarrow nx; ix++)
01456
         met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
} else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR
01457
01458
                      || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
01460
           NC(nc_get_var_float(ncid, varid, help));
01461
            for (iy = 0; iy < met->ny; iy++)
              for (ix = 0; ix < met->nx; ix++)
01462
               met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
01463
01464
         } else
01465
           for (ix = 0; ix < met->nx; ix++)
01466
              for (iy = 0; iy < met->ny; iy++)
01467
                met->ps[ix][iy] = met->p[0];
01468
01469
         /* Create periodic boundary conditions... */
01470
         read met periodic (met);
01471
01472
         /* Calculate geopotential heights... */
01473
         read_met_geopot(ctl, met);
01474
01475
         /* Calculate potential vorticity... */
01476
         read_met_pv(met);
```

```
01477
01478
        /* Calculate tropopause pressure... */
01479
        read_met_tropo(ctl, met);
01480
01481
       /* Downsampling... */
       read_met_sample(ctl, met);
01482
01483
01484
        /* Close file...
01485
       NC(nc_close(ncid));
01486 }
```



## 5.21.2.24 void read\_met\_extrapolate ( met\_t \* met )

Extrapolate meteorological data at lower boundary.

Definition at line 1490 of file libtrac.c.

```
01491
01492
01493
         int ip, ip0, ix, iy;
01494
         /* Loop over columns... */
for (ix = 0; ix < met->nx; ix++)
  for (iy = 0; iy < met->ny; iy++) {
01495
01496
01497
01498
01499
               /* Find lowest valid data point... */
01500
               for (ip0 = met->np - 1; ip0 >= 0; ip0--)
                 if (!gsl_finite(met->t[ix][iy][ip0])
01501
                      || !gsl_finite(met->u[ix][iy][ip0])
|| !gsl_finite(met->v[ix][iy][ip0])
01502
01503
01504
                      || !gsl_finite(met->w[ix][iy][ip0]))
01505
01506
              /* Extrapolate... */
for (ip = ip0; ip >= 0; ip--) {
    met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
01507
01508
01509
01510
                met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
                met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
                met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
01512
                met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
01513
                met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
01514
01515
01516
            }
01517 }
```

```
5.21.2.25 void read_met_geopot ( ctl_t * ctl, met_t * met )
```

Calculate geopotential heights.

Definition at line 1521 of file libtrac.c.

```
01523
01524
01525
        static double topo_lat[EY], topo_lon[EX], topo_z[EX][EY];
01526
        static int init, topo nx = -1, topo ny;
01527
01528
01529
        FILE *in;
01530
01531
        char line[LEN];
01532
01533
        double data[30], lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;
01534
01535
        float help[EX][EY];
01536
        int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;
01537
01538
01539
        /* Initialize geopotential heights... */
01540
        for (ix = 0; ix < met->nx; ix++)
         for (iy = 0; iy < met >nx; iy++)
for (ip = 0; ip < met ->ny; ip++)
01541
01542
01543
              met->z[ix][iy][ip] = GSL_NAN;
01544
        /* Check filename... */
if (ctl->met_geopot[0] == '-')
01545
01546
01547
          return;
01548
        /* Read surface geopotential... */
01550
        if (!init) {
01551
          init = 1;
01552
          /* Write info... */
01553
01554
          printf("Read surface geopotential: %s\n", ctl->met_geopot);
01555
01556
          if (!(in = fopen(ctl->met_geopot, "r")))
    ERRMSG("Cannot open file!");
01557
01558
01559
01560
          /* Read data... */
          /* Read Gata... "/
while (fgets(line, LEN, in))
  if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rz) == 3) {
01561
01562
01563
               if (rlon != rlon_old) {
                if ((++topo_nx) >= EX)
    ERRMSG("Too many longitudes!");
01564
01565
                 topo_ny = 0;
01566
01567
               rlon_old = rlon;
01568
01569
               topo_lon[topo_nx] = rlon;
               topo_lat[topo_ny] = rlat;
01570
              topo_z[topo_nx][topo_ny] = rz;
if ((++topo_ny) >= EY)
01571
01572
01573
                 ERRMSG("Too many latitudes!");
01574
01575
          if ((++topo_nx) >= EX)
01576
            ERRMSG("Too many longitudes!");
01577
01578
          /* Close file... */
01579
          fclose(in);
01581
           /* Check grid spacing... */
01582
          || fabs(met->lat[0] - met->lat[1]) != fabs(topo_lat[0] - topo_lat[1]))
01583
01584
             printf("Warning: Grid spacing does not match!\n");
01585
01586
01587
         /\star Apply hydrostatic equation to calculate geopotential heights... \star/
01588
        for (ix = 0; ix < met->nx; ix++)
01589
          for (iy = 0; iy < met->ny; iy++) {
01590
             /* Get surface height... */
01591
01592
             lon = met->lon[ix];
01593
            if (lon < topo_lon[0])</pre>
01594
               lon += 360;
            else if (lon > topo_lon[topo_nx - 1])
  lon -= 360;
01595
01596
01597
            lat = met->lat[iy];
01598
            tx = locate(topo_lon, topo_nx, lon);
01599
            ty = locate(topo_lat, topo_ny, lat);
```

```
z0 = LIN(topo_lon[tx], topo_z[tx][ty],
                     topo_lon[tx + 1], topo_z[tx + 1][ty], lon);
01601
            z1 = LIN(topo_lon[tx], topo_z[tx][ty + 1],
topo_lon[tx + 1], topo_z[tx + 1][ty + 1], lon);
01602
01603
01604
            z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);
01605
01606
            /* Find surface pressure level... */
            ip0 = locate(met->p, met->np, met->ps[ix][iy]);
01607
01608
            01609
01610
01611
01612
01613
            /* Upper part of profile... */
01614
            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]));
01615
01616
            for (ip = ip0 + 2; ip < met->np; ip++)
01617
             met->z[ix][iy][ip]
01618
01619
               = (float) (met->z[ix][iy][ip - 1] + RI / MA / G0
01620
                           * 0.5 * (met->t[ix][iy][ip - 1] + met->t[ix][iy][ip])
01621
                            * log(met->p[ip - 1] / met->p[ip]));
01622
         }
01623
        /* Smooth fields... */
01624
01625
       for (ip = 0; ip < met->np; ip++) {
01626
          /* Median filter...
01627
01628
          for (ix = 0; ix < met->nx; ix++)
01629
            for (iy = 0; iy < met->nx; iy++) {
01630
             n = 0;
              for (ix2 = ix - 2; ix2 \le ix + 2; ix2++) {
01631
01632
               ix3 = ix2;
01633
                if (ix3 < 0)
01634
                  ix3 += met->nx;
                if (ix3 >= met->nx)
01635
                  ix3 -= met -> nx;
01636
                for (iy2 = GSL_MAX(iy - 2, 0); iy2 <= GSL_MIN(iy + 2, met->ny - 1);
01637
                     iy2++)
01638
01639
                  if (gsl_finite(met->z[ix3][iy2][ip])) {
01640
                    data[n] = met \rightarrow z[ix3][iy2][ip];
01641
                    n++;
                  }
01642
01643
              if (n > 0) {
01644
01645
                gsl_sort(data, 1, (size_t) n);
01646
                help[ix][iy] = (float)
01647
                  gsl_stats_median_from_sorted_data(data, 1, (size_t) n);
              } else
01648
01649
                help[ix][iy] = GSL_NAN;
01650
            }
01651
01652
          /* Copy data... */
         for (ix = 0; ix < met->nx; ix++)
  for (iy = 0; iy < met->nx; iy++)
01653
01654
              met->z[ix][iy][ip] = help[ix][iy];
01655
01656
01657 }
```

5.21.2.26 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 1661 of file libtrac.c.

```
01667
                   {
01668
       static float help[EX * EY * EP];
01669
01670
01671
       int ip, ix, iy, varid;
01672
        /* Check if variable exists... */
01673
01674
        if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
01675
         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
01676
           return:
01677
01678
       /* Read data... */
```

```
NC(nc_get_var_float(ncid, varid, help));
01680
01681
           /* Copy and check data... */
01682
           for (ix = 0; ix < met->nx; ix++)
            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>
01683
01684
01685
01686
01687
                     dest[ix][iy][ip] *= scl;
01688
                   else
01689
                      dest[ix][iy][ip] = GSL_NAN;
01690
01691 }
```

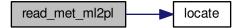
5.21.2.27 void read\_met\_ml2pl (  $ctl_t * ctl$ ,  $met_t * met$ , float var[EX][EY][EP] )

Convert meteorological data from model levels to pressure levels.

Definition at line 1695 of file libtrac.c.

```
01698
01699
01700
        double aux[EP], p[EP], pt;
01701
01702
        int ip, ip2, ix, iy;
01703
01704
         /* Loop over columns... ∗/
01705
        for (ix = 0; ix < met->nx; ix++)
01706
          for (iy = 0; iy < met->ny; iy++) {
01707
01708
             /* Copy pressure profile... */
for (ip = 0; ip < met->np; ip++)
01709
01710
               p[ip] = met->pl[ix][iy][ip];
01711
             /* 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]))
01712
01713
01714
01716
                 pt = p[0];
01717
               else if ((pt > p[met->np - 1] && p[1] > p[0])
01718
                          || (pt < p[met->np - 1] && p[1] < p[0]))
              01719
01720
01721
01722
01723
01724
             /* Copy data... */
for (ip = 0; ip < ctl->met_np; ip++)
01725
01726
               var[ix][iy][ip] = (float) aux[ip];
01727
01728
01729 }
```

Here is the call graph for this function:



```
5.21.2.28 void read_met_periodic ( met_t * met )
```

Create meteorological data with periodic boundary conditions.

Definition at line 1733 of file libtrac.c.

```
01734
01735
01736
           int ip, iv;
01738
           /* Check longitudes... */
01739
           if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
01740
                            + met -> lon[1] - met -> lon[0] - 360) < 0.01))
01741
01742
01743
           /* Increase longitude counter... */
           if ((++met->nx) > EX)
01744
01745
              ERRMSG("Cannot create periodic boundary conditions!");
01746
          /* Set longitude... */
met->lon[met->nx - 2] + met->lon[1] - met->
01747
01748
        lon[0];
01749
01750
            /* Loop over latitudes and pressure levels... */
           for (iy = 0; iy < met->ny; iy++)
  for (ip = 0; ip < met->np; ip++) {
01751
01752
                 met->ps[met->nx - 1][iy] = met->ps[0][iy];
met->pt[met->nx - 1][iy] = met->pt[0][iy];
met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
01753
01754
01755
                met->z[met->nx - 1][iy][ip] = met->z[o][iy][ip];

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

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

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

met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
01756
01757
01758
01759
                met->pv[met->nx - 1][iy][ip] = met->pv[0][iy][ip];
met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
01760
01761
01762
                 met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01763
01764 }
```

# 5.21.2.29 void read\_met\_pv ( met\_t \* met )

Calculate potential vorticity.

Definition at line 1768 of file libtrac.c.

```
01769
01770
01771
        double c0, c1, cr, dx, dy, dp, dtdx, dvdx, dtdy, dudy, dtdp, dudp, dvdp,
01772
          latr, vort, pows[EP];
01774
        int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
01775
        /* Set powers... */
for (ip = 0; ip < met->np; ip++)
01776
01777
         pows[ip] = pow(1000. / met->p[ip], 0.286);
01778
01779
01780
         /* Loop over grid points... */
01781
        for (ix = 0; ix < met->nx; ix++) {
01782
          /* Set indices... */
ix0 = GSL_MAX(ix - 1, 0);
01783
01784
01785
          ix1 = GSL_MIN(ix + 1, met -> nx - 1);
01786
01787
           /* Loop over grid points... */
01788
           for (iy = 0; iy < met->ny; iy++) {
01789
             /* Set indices... */
iy0 = GSL_MAX(iy - 1, 0);
01790
01791
             iy1 = GSL_MIN(iy + 1, met->ny - 1);
01792
01793
01794
             /* Set auxiliary variables... */
             latr = GSL_MIN(GSL_MAX(met->lat[iy], -89.), 89.);
01795
01796
             dx = 1000. * deg2dx(met->lon[ix1] - met->lon[ix0], latr);
dy = 1000. * deg2dy(met->lat[iy1] - met->lat[iy0]);
01797
01798
             c0 = cos(met->lat[iy0] / 180. * M_PI);
01799
             c1 = cos(met->lat[iy1] / 180. * M_PI);
```

```
cr = cos(latr / 180. * M_PI);
             vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
01801
01802
01803
              /\star Loop over grid points... \star/
01804
             for (ip = 0; ip < met->np; ip++) {
01805
01806
                /* Set indices... */
01807
                ip0 = GSL\_MAX(ip - 1, 0);
                ip1 = GSL_MIN(ip + 1, met->np - 1);
01808
01809
                /\star Set auxiliary variables... \star/
01810
01811
                dp = 100. * (met->p[ip1] - met->p[ip0]);
01812
01813
                /* 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;
01814
01815
01816
                /* Get gradients in latitude... */
01817
                dtdy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
01818
                dudy = (met -> u[ix][iy1][ip] * c1 - met -> u[ix][iy0][ip] * c0) / dy;
01819
01820
01821
                /* Get gradients in pressure... */
01822
                dtdp =
                (met->t[ix][iy][ip1] * pows[ip1] -
  met->t[ix][iy][ip0] * pows[ip0]) / dp;
dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / dp;
01823
01824
01825
01826
                dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / dp;
01827
01828
                /* Calculate PV... */
                met->pv[ix][iy][ip] = (float)
(1e6 * G0 *
01829
01830
01831
                    (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
01832
01833
01834
        }
01835 }
```

Here is the call graph for this function:

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

Downsampling of meteorological data.

Definition at line 1839 of file libtrac.c.

```
01841
                           {
01842
01843
         met_t *help;
01844
01845
         float w, wsum;
01846
01847
         int ip, ip2, ix, ix2, iy, iy2;
01848
01849
         /* Check parameters... */
01850
          if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1)</pre>
01851
            return;
01852
          /* Allocate... */
01853
01854
         ALLOC(help, met_t, 1);
01856
          /* Copy data... */
01857
         help->nx = met->nx;
         help->ny = met->ny;
01858
          help->np = met->np;
01859
         memcpy(help->lon, met->lon, sizeof(met->lon));
memcpy(help->lat, met->lat, sizeof(met->lat));
01860
01861
01862
          memcpy(help->p, met->p, sizeof(met->p));
01863
01864
          /* Smoothing... */
          for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
01865
           for (iy = 0; iy < met >ny; iy += ctl >met_dy) {
   for (ip = 0; ip < met >np; ip += ctl ->met_dp) {
    for (ip = 0; ip < met ->np; ip += ctl ->met_dp) {

01866
01867
01868
                  help \rightarrow ps[ix][iy] = 0;
                  help \rightarrow pt[ix][iy] = 0;
01869
01870
                  help \rightarrow z[ix][iy][ip] = 0;
                  help->t[ix][iy][ip] = 0;
01871
                 help->u[ix][iy][ip] = 0;
help->v[ix][iy][ip] = 0;
01872
01873
01874
                 help->w[ix][iy][ip] = 0;
```

```
help \rightarrow pv[ix][iy][ip] = 0;
                   help->h2o[ix][iy][ip] = 0;
01876
01877
                   help \rightarrow 03[ix][iy][ip] = 0;
01878
                   wsum = 0;
                   for (ix2 = GSL_MAX(ix - ctl->met_sx + 1, 0);
    ix2 <= GSL_MIN(ix + ctl->met_sx - 1, met->nx - 1); ix2++)
    for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
01879
01880
01881
                        iv2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
iv2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iv2++)
for (ip2 = GSL_MIX(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)
    * (float) (1.0 - fabs(ip - ip2) / ctl->met_sp);
01882
01883
01884
01885
01886
01887
                           help \rightarrow ps[ix][iy] += w * met \rightarrow ps[ix2][iy2];
01888
                           help->pt[ix][iy] += w * met->pt[ix2][iy2];
01889
                          help->z[ix][iy][ip] += w * met->z[ix2][iy2][ip2];
help->t[ix][iy][ip] += w * met->t[ix2][iy2][ip2];
help->u[ix][iy][ip] += w * met->u[ix2][iy2][ip2];
01890
01891
01892
                          help->v[ix][iy][ip] += w * met->v[ix2][iy2][ip2];
01893
                           help \rightarrow w[ix][iy][ip] += w * met \rightarrow w[ix2][iy2][ip2];
01894
01895
                           help \rightarrow pv[ix][iy][ip] += w * met \rightarrow pv[ix2][iy2][ip2];
01896
                           help->h2o[ix][iy][ip] += w * met->h2o[ix2][iy2][ip2];
                           \label{eq:help-o3[ix][iy][ip] += w * met->o3[ix2][iy2][ip2];} \\
01897
01898
                           wsum += w;
01899
01900
                   help->ps[ix][iy] /= wsum;
01901
                   help->pt[ix][iy] /= wsum;
01902
                   help->t[ix][iy][ip] /= wsum;
                   help->z[ix][iy][ip] /= wsum;
01903
                   help->u[ix][iy][ip] /= wsum;
01904
                   help->v[ix][iy][ip] /= wsum;
01905
01906
                   help->w[ix][iy][ip] /= wsum;
01907
                   help->pv[ix][iy][ip] /= wsum;
01908
                   help->h2o[ix][iy][ip] /= wsum;
01909
                   help->03[ix][iy][ip] /= wsum;
01910
01911
            }
01912
01913
01914
           /* Downsampling... */
          met->nx = 0;
for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
01915
01916
            met->lon[met->nx] = help->lon[ix];
01917
             met->ny = 0;
for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
01918
01919
01920
                met->lat[met->ny] = help->lat[iy];
01921
                met->ps[met->nx][met->ny] = help->ps[ix][iy];
                met->pt[met->nx][met->ny] = help->pt[ix][iy];
01922
01923
                met->np = 0;
                for (ip = 0; ip < help->np; ip += ctl->met_dp) {
01924
01925
                  met \rightarrow p[met \rightarrow np] = help \rightarrow p[ip];
01926
                   met->z[met->nx][met->ny][met->np] = help->z[ix][iy][ip];
01927
                   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];
met->v[met->nx][met->ny][met->np] = help->v[ix][iy][iy];
01928
01929
                   met->w[met->nx] [met->ny] [met->np] = help->w[ix][iy][ip];
01930
                  met->pv[met->nx][met->ny][met->np] = help->pv[ix][iy][ip];
01932
                   met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
01933
                   met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
01934
                   met->np++;
01935
01936
                met->ny++;
01937
01938
             met->nx++;
01939
01940
           /* Free... */
01941
          free(help);
01942
01943 }
```

## 5.21.2.31 void read\_met\_tropo ( ctl\_t \* ctl, met\_t \* met )

Calculate tropopause pressure.

Definition at line 1947 of file libtrac.c.

```
01949 {
01950
01951 gsl_interp_accel *acc;
01952
```

```
gsl_spline *spline;
01954
        double p2[400], pv[400], pv2[400], t[400], t2[400], th[400], th2[400],
01955
01956
          z[400], z2[400];
01957
01958
        int found, ix, iv, iz, iz2;
01959
01960
        /* Allocate... */
01961
         acc = gsl_interp_accel_alloc();
01962
         spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) met->np);
01963
01964
         /\star Get altitude and pressure profiles... \star/
        for (iz = 0; iz < met->np; iz++)
01965
01966
           z[iz] = Z(met->p[iz]);
01967
         for (iz = 0; iz <= 170; iz++) {
         z2[iz] = 4.5 + 0.1 * iz;
01968
           p2[iz] = P(z2[iz]);
01969
01970
01971
01972
         /* Do not calculate tropopause... */
01973
         if (ctl->met_tropo == 0)
           for (ix = 0; ix < met->nx; ix++)
01974
             for (iy = 0; iy < met->ny; iy++)
01975
01976
               met->pt[ix][iy] = GSL_NAN;
01977
01978
        /* Use tropopause climatology... */
01979
        else if (ctl->met_tropo == 1)
01980
         for (ix = 0; ix < met->nx; ix++)
01981
             for (iy = 0; iy < met->ny; iy++)
               met->pt[ix][iy] = clim_tropo(met->time, met->lat[iy]);
01982
01983
01984
        /* Use cold point... */
01985
        else if (ctl->met_tropo == 2) {
01986
           /* Loop over grid points... */
for (ix = 0; ix < met->nx; ix++)
  for (iy = 0; iy < met->ny; iy++) {
01987
01988
01989
01990
01991
                /* Interpolate temperature profile... */
                for (iz = 0; iz < met->np; iz++)
  t[iz] = met->t[ix][iy][iz];
01992
01993
                gsl_spline_init(spline, z, t, (size_t) met->np);
for (iz = 0; iz <= 170; iz++)</pre>
01994
01995
01996
                 t2[iz] = gsl_spline_eval(spline, z2[iz], acc);
01997
01998
                /* Find minimum...
                iz = (int) gsl_stats_min_index(t2, 1, 171);
if (iz <= 0 || iz >= 170)
01999
02000
                 met->pt[ix][iy] = GSL_NAN;
02001
02002
                else
02003
                  met \rightarrow pt[ix][iy] = p2[iz];
02004
02005
02006
02007
        /* Use WMO definition... */
02008
        else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
02009
02010
           /* Loop over grid points... */
02011
           for (ix = 0; ix < met->nx; ix++)
02012
             for (iy = 0; iy < met->ny; iy++) {
02013
02014
                /\star Interpolate temperature profile... \star/
                for (iz = 0; iz < met->np; iz++)
  t[iz] = met->t[ix][iy][iz];
02015
02016
02017
                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>
02018
02019
02020
                /* Find 1st tropopause... */
met->pt[ix][iy] = GSL_NAN;
02021
                for (iz = 0; iz <= 140; iz++) {
  found = 1;
02023
02024
                  for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
  if (1000. * G0 / RA * log(t2[iz2] / t2[iz])</pre>
02025
02026
                         / log(p2[iz2] / p2[iz]) > 2.0) {
02027
02028
                       found = 0;
02029
                       break;
02030
02031
                  if (found) {
                    if (iz > 0 && iz < 140)
02032
                      met->pt[ix][iy] = p2[iz];
02033
02034
                    break;
02035
02036
02037
                /* Find 2nd tropopause... */
02038
02039
                if (ctl->met_tropo == 4) {
```

```
met->pt[ix][iy] = GSL_NAN;
02041
                for (; iz <= 140; iz++) {
02042
                  found = 1;
                  02043
02044
02045
                      found = 0;
02047
                      break;
02048
02049
                  if (found)
02050
                    break;
02051
02052
                for (; iz <= 140; iz++) {</pre>
02053
                 found = 1;
02054
                  for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)</pre>
02055
                   if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
02056
                        / \log(p2[iz2] / p2[iz]) > 2.0) {
02057
                      found = 0;
02058
                      break;
02060
                  if (found) {
02061
                    if (iz > 0 && iz < 140)
02062
                      met \rightarrow pt[ix][iy] = p2[iz];
02063
                    break;
02064
                  }
02065
                }
02066
              }
02067
            }
02068
02069
02070
       /* Use dynamical tropopause... */
       else if (ctl->met_tropo == 5) {
02072
02073
          /\star Loop over grid points... \star/
02074
         for (ix = 0; ix < met->nx; ix++)
02075
            for (iy = 0; iy < met->ny; iy++) {
02076
              /* Interpolate potential vorticity profile... */
02078
              for (iz = 0; iz < met->np; iz++)
02079
                pv[iz] = met->pv[ix][iy][iz];
02080
              gsl_spline_init(spline, z, pv, (size_t) met->np);
02081
              for (iz = 0; iz <= 160; iz++)</pre>
               pv2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02082
02083
              /* Interpolate potential temperature profile... */
              for (iz = 0; iz < met->np; iz++)
02085
02086
               th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
02087
              gsl_spline_init(spline, z, th, (size_t) met->np);
02088
              for (iz = 0; iz <= 160; iz++)</pre>
               th2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02089
02090
02091
              /\star Find dynamical tropopause 3.5 PVU + 380 K \star/
02092
              met->pt[ix][iy] = GSL_NAN;
              for (iz = 0; iz <= 160; iz++)

if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
02093
02094
02095
                 if (iz > 0 && iz < 160)
                   met->pt[ix][iy] = p2[iz];
02097
                  break;
02098
02099
            }
02100
       }
02101
02102
       else
02103
         ERRMSG("Cannot calculate tropopause!");
02104
        /* Free... */
02105
02106
       gsl_spline_free(spline);
02107
       gsl_interp_accel_free(acc);
02108 }
```

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

Read a control parameter from file or command line.

Definition at line 2112 of file libtrac.c.

```
02119
02120
02121
        FILE *in = NULL;
02122
02123
        char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
02124
          msg[2 * LEN], rvarname[LEN], rval[LEN];
02125
02126
02127
02128
         /* Open file... */
        if (filename[strlen(filename) - 1] != '-')
02129
         if (!(in = fopen(filename, "r")))
02130
            ERRMSG("Cannot open file!");
02131
02132
02133
        /* Set full variable name... */
02134
        if (arridx >= 0) {
         sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
02135
02136
02137
        } else {
          sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
02138
02139
02140
02141
02142
        /* Read data... */
02143
        if (in != NULL)
         while (fgets(line, LEN, in))
02144
02145
            if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
              if (strcasecmp(rvarname, fullname1) == 0 ||
02146
02147
                   strcasecmp(rvarname, fullname2) == 0) {
02148
                 contain = 1;
02149
                 break:
02150
               }
02151
        for (i = 1; i < argc - 1; i++)</pre>
02152
         if (strcasecmp(argv[i], fullname1) == 0 ||
             strcasecmp(argv[i], fullname2) == 0) {
sprintf(rval, "%s", argv[i + 1]);
02153
02154
02155
             contain = 1;
02156
             break;
02157
02158
02159
        /* Close file... */
        if (in != NULL)
02160
02161
         fclose(in);
02162
02163
        /* Check for missing variables... */
02164
        if (!contain) {
         if (strlen(defvalue) > 0)
   sprintf(rval, "%s", defvalue);
02165
02166
          else {
02167
02168
            sprintf(msg, "Missing variable %s!\n", fullname1);
02169
             ERRMSG (msg);
02170
02171
02172
02173
        /* Write info... */
       printf("%s = %s\n", fullname1, rval);
02174
02176
        /* Return values... */
02177
        if (value != NULL)
          sprintf(value, "%s", rval);
02178
02179
        return atof(rval);
02180 }
```

5.21.2.33 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 2184 of file libtrac.c.

```
02192
                         {
02193
02194
        struct tm t0, t1;
02195
02196
        t0.tm_year = 100;
02197
         t0.tm_mon = 0;
        t0.tm_mday = 1;
t0.tm_hour = 0;
02198
02199
02200
        t0.tm_min = 0;
02201
        t0.tm\_sec = 0;
```

```
02203    t1.tm_year = year - 1900;
02204    t1.tm_mon = mon - 1;
02205    t1.tm_mday = day;
02206    t1.tm_hour = hour;
02207    t1.tm_min = min;
02208    t1.tm_sec = sec;
02209
02210    *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
02211 }
```

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

Measure wall-clock time.

Definition at line 2215 of file libtrac.c.

```
02218
02219
02220
        static double starttime[NTIMER], runtime[NTIMER];
02221
02222
        /* Check id... */
        if (id < 0 || id >= NTIMER)
02224
          ERRMSG("Too many timers!");
02225
02226
        /* Start timer... */
02227
        if (mode == 1) {
        if (starttime[id] <= 0)</pre>
02228
            starttime[id] = omp_get_wtime();
02230
02231
            ERRMSG("Timer already started!");
02232
02233
        /* Stop timer... */
else if (mode == 2) {
02234
02235
        if (starttime[id] > 0) {
02236
02237
           runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
02238
            starttime[id] = -1;
02239
02240 }
02241
02242
        /* Print timer... */
       else if (mode == 3) {
    printf("%s = %.3f s\n", name, runtime[id]);
02243
02244
          runtime[id] = 0;
02245
        }
02246
02247 }
```

5.21.2.35 void write\_atm ( const char \* filename, ctl\_t \* ctl, atm\_t \* atm, double t )

Write atmospheric data.

Definition at line 2251 of file libtrac.c.

```
02255
                   {
02257
       FILE *in, *out;
02258
02259
        char line[LEN];
02260
02261
        double r, t0, t1;
02262
02263
        int ip, iq, year, mon, day, hour, min, sec;
02264
02265
        /\star Set time interval for output... \star/
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
02266
02267
02268
02269
        /* Write info... */
02270
        printf("Write atmospheric data: %s\n", filename);
02271
        /* Write ASCII data... */
02272
02273
        if (ctl->atm_type == 0) {
02274
          /* Check if gnuplot output is requested... */
```

```
02276
          if (ctl->atm_gpfile[0] != '-') {
02277
            /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
02278
02279
             ERRMSG("Cannot create pipe to gnuplot!");
02280
02281
            02283
02284
02285
            /* Set time string... */
            jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02286
02287
                    year, mon, day, hour, min);
02288
02289
02290
            /\star Dump gnuplot file to pipe... \star/
            if (!(in = fopen(ctl->atm_gpfile, "r")))
    ERRMSG("Cannot open file!");
02291
02292
            while (fgets(line, LEN, in))
fprintf(out, "%s", line);
02293
02295
            fclose(in);
02296
02297
02298
         else {
02299
02300
            /* Create file... */
02301
            if (!(out = fopen(filename, "w")))
02302
              ERRMSG("Cannot create file!");
02303
02304
          /* Write header... */
02305
02306
          fprintf(out,
02307
                  "# $1 = time [s]\n"
02308
                  "# $2 = altitude [km] \n"
                  "# $3 = longitude [deg] \n" "# $4 = latitude [deg] \n");
02309
         02310
02311
02312
02313
02314
02315
          /* Write data... */
02316
          for (ip = 0; ip < atm->np; ip++) {
02317
            /\star Check time... \star/
02318
            if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
02319
02320
             continue;
02321
           02322
02323
02324
02325
02326
02327
              fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02328
02329
           fprintf(out, "\n");
02330
02331
02332
          /* Close file... */
02333
         fclose(out);
02334
02335
02336
       /* Write binary data... */
02337
       else if (ctl->atm_type == 1) {
02338
02339
          /* Create file... */
          if (!(out = fopen(filename, "w")))
02340
           ERRMSG("Cannot create file!");
02341
02342
          /* Write data... */
02343
02344
          FWRITE(&atm->np, int,
02345
                 1,
02346
                 out);
02347
          FWRITE(atm->time, double,
02348
                  (size_t) atm->np,
02349
                 out);
02350
         FWRITE(atm->p, double,
02351
                  (size_t) atm->np,
02352
                 out);
02353
          FWRITE(atm->lon, double,
02354
                  (size_t) atm->np,
02355
                 out):
         FWRITE(atm->lat, double,
02356
02357
                   (size_t) atm->np,
02358
                 out);
02359
          for (iq = 0; iq < ctl->nq; iq++)
02360
           FWRITE(atm->q[iq], double,
02361
                    (size_t) atm->np,
02362
                   out);
```



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

Write CSI data.

Definition at line 2375 of file libtrac.c.

```
02380
02381
        static FILE *in, *out;
02382
02383
        static char line[LEN];
02384
02385
        static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
02386
           rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
02387
02388
        static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
02389
02390
        /* Init... */
02391
         if (t == ctl->t_start) {
02392
02393
           /\star Check quantity index for mass... \star/
02394
           if (ctl->qnt_m < 0)
   ERRMSG("Need quantity mass!");</pre>
02395
02396
02397
            /\star Open observation data file... \star/
           printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
if (!(in = fopen(ctl->csi_obsfile, "r")))
    ERRMSG("Cannot open file!");
02398
02399
02400
02401
02402
           /* Create new file... */
           printf("Write CSI data: %s\n", filename);
if (!(out = fopen(filename, "w")))
02403
02404
02405
              ERRMSG("Cannot create file!");
02406
           /* Write header... */
02407
02408
           fprintf(out,
02409
                     "# $1 = time [s] \n"
02410
                     "# $2 = number of hits (cx)\n"
02411
                     "# $3 = number of misses (cy) \n"
                     "# $4 = number of false alarms (cz)\n"
02412
                     "# $5 = number of observations (cx + cy) \n"
"# $6 = number of forecasts (cx + cz) \n"
02413
02414
                     "# $7 = bias (forecasts/observations) [%%]\n"
02415
02416
                     "# $8 = probability of detection (POD) [%%]\n"
                     "# $9 = false alarm rate (FAR) [%%] \n"
02417
02418
                     "# $10 = critical success index (CSI) [%%]\n\n");
02419
02420
02421
        /* Set time interval... */
02422
        t0 = t - 0.5 * ct1->dt_mod;
```

```
t1 = t + 0.5 * ctl->dt_mod;
02424
02425
         /* Initialize grid cells... */
02426
        for (ix = 0; ix < ctl->csi_nx; ix++)
02427
          for (iy = 0; iy < ctl->csi_ny; iy++)
  for (iz = 0; iz < ctl->csi_nz; iz++)
02428
               modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
02430
02431
         /* Read observation data... */
02432
        while (fgets(line, LEN, in)) {
02433
02434
           /* Read data... */
if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02435
02436
02437
             continue;
02438
           /* Check time... */
02439
02440
          if (rt < t0)</pre>
02441
             continue;
           if (rt > t1)
02442
02443
            break;
02444
02445
           /* Calculate indices... */
          ix = (int) ((rlon - ctl->csi_lon0)
02446
02447
                         / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
           iy = (int) ((rlat - ctl->csi_lat0)
02449
                         / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02450
           iz = (int) ((rz - ctl->csi_z0)
02451
                        / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02452
02453
           /* Check indices... */
02454
           if (ix < 0 || ix >= ctl->csi_nx ||
02455
               iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02456
             continue;
02457
02458
           /* Get mean observation index... */
02459
           obsmean[ix][iy][iz] += robs;
02460
          obscount[ix][iy][iz]++;
02461
02462
02463
         /* Analyze model data... ∗/
02464
        for (ip = 0; ip < atm->np; ip++) {
02465
02466
           /* Check time... */
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
02467
02468
             continue;
02469
02470
           /* Get indices... */
02471
          ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
                         / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02472
           iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
02474
                         / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
          02475
02476
02477
02478
           /* Check indices... */
          if (ix < 0 || ix >= ctl->csi_nx ||
02480
               iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02481
02482
           /\star Get total mass in grid cell... \star/
02483
          modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02484
02485
02486
02487
         /* Analyze all grid cells... */
02488
        for (ix = 0; ix < ctl->csi_nx; ix++)
02489
          for (iy = 0; iy < ctl->csi_ny; iy++)
  for (iz = 0; iz < ctl->csi_nz; iz++) {
02490
02491
02492
               /* Calculate mean observation index... */
02493
               if (obscount[ix][iy][iz] > 0)
02494
                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
02495
               /* Calculate column density... */
02496
               if (modmean[ix][iy][iz] > 0) {
  dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
02497
02498
                 dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
02499
                 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);
02500
02501
02502
02503
02504
02505
02506
               /* Calculate CSI... */
02507
               if (obscount[ix][iy][iz] > 0) {
                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
    modmean[ix][iy][iz] >= ctl->csi_modmin)
02508
02509
```

```
cx++;
02511
                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02512
                            modmean[ix][iy][iz] < ctl->csi_modmin)
02513
                   cv++;
02514
                 02515
02516
                    cz++;
02517
02518
            }
02519
        /* Write output... */
if (fmod(t, ctl->csi_dt_out) == 0) {
02520
02521
02522
          /* Write... */    fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
02523
02524
                    (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,
02525
02526
02527
02528
                    (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
02530
02531
           /* Set counters to zero... */
02532
          cx = cy = cz = 0;
02533
02534
02535
        /* Close file... */
        if (t == ctl->t_stop)
02536
02537
          fclose(out);
02538 }
```

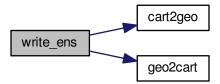
5.21.2.37 void write\_ens ( const char \* filename, ctl\_t \* ctl, atm\_t \* atm, double t )

Write ensemble data.

Definition at line 2542 of file libtrac.c.

```
02547
02548
        static FILE *out;
02549
02550
        static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
02551
         t0, t1, x[NENS][3], xm[3];
02552
02553
        static int ip, iq;
02554
02555
        static size_t i, n;
02556
02557
        /* Init... */
02558
        if (t == ctl->t_start) {
02559
02560
          /* Check quantities... */
02561
         if (ctl->qnt_ens < 0)
    ERRMSG("Missing ensemble IDs!");</pre>
02562
02563
02564
          /\star Create new file... \star/
          printf("Write ensemble data: %s\n", filename);
if (!(out = fopen(filename, "w")))
02565
02566
02567
           ERRMSG("Cannot create file!");
02568
02569
          /* Write header... */
          fprintf(out,
02571
                   "# $1 = time [s] \n"
02572
                   "# $2 = altitude [km] \n"
                  "# \$3 = longitude [deg]\n" "# \$4 = latitude [deg]\n");
02573
          for (iq = 0; iq < ctl->nq; iq++)
fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
02574
02575
                     ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02576
          02577
02578
02579
02580
02581
02582
02583
        /* Set time interval... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
02584
02585
02586
02587
        /* Init...
02588
        ens = GSL_NAN;
02589
        n = 0;
```

```
02590
02591
          /* Loop over air parcels... */
02592
         for (ip = 0; ip < atm->np; ip++) {
02593
02594
            /* Check time... */
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
02595
02596
             continue;
02597
02598
            /* Check ensemble id... */
02599
            if (atm->q[ctl->qnt_ens][ip] != ens) {
02600
02601
              /* Write results... */
02602
              if (n > 0) {
02603
02604
                /* Get mean position... */
                for (i = 0; i < n; i++) {
    xm[0] += x[i][0] / (double) n;
    xm[1] += x[i][1] / (double) n;</pre>
02605
02606
02607
02608
02609
                  xm[2] += x[i][2] / (double) n;
02610
02611
                cart2geo(xm, &dummy, &lon, &lat);
                02612
02613
                          lat):
02614
02615
                 /* Get quantity statistics... */
                for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02616
02617
02618
                   fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02619
                for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02620
02621
02622
                   fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02623
02624
                fprintf(out, " lu\n", n);
02625
02626
02627
              /* Init new ensemble... */
02628
              ens = atm->q[ctl->qnt_ens][ip];
02629
              n = 0;
02630
02631
            /* Save data... */
02632
02633
           p[n] = atm -> p[ip];
            geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
02634
02635
            for (iq = 0; iq < ctl->nq; iq++)
           q[iq][n] = atm->q[iq][ip];
if ((++n) >= NENS)
    ERRMSG("Too many data points!");
02636
02637
02638
02639
02640
02641
          /* Write results... */
02642
         if (n > 0) {
02643
02644
            /* Get mean position... */
           xm[0] = xm[1] = xm[2] = 0;
for (i = 0; i < n; i++) {
02645
02646
              xm[0] += x[i][0] / (double) n;
xm[1] += x[i][1] / (double) n;
xm[2] += x[i][2] / (double) n;
02647
02648
02649
02650
           cart2geo(xm, &dummy, &lon, &lat);
fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
02651
02652
02653
02654
            /* Get quantity statistics... */
            for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02655
02656
              fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02657
02658
            for (iq = 0; iq < ctl->nq; iq++) {
   fprintf(out, " ");
   fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02659
02660
02661
02662
            fprintf(out, " lu\n", n);
02663
02664
02665
02666
          /* Close file... */
02667
         if (t == ctl->t_stop)
02668
            fclose(out);
02669 }
```



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

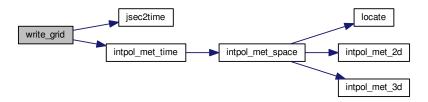
Write gridded data.

Definition at line 2673 of file libtrac.c.

```
02679
                       {
02681
         FILE *in, *out;
02682
02683
         char line[LEN];
02684
02685
         static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
02686
            area, rho_air, press, temp, cd, vmr, t0, t1, r;
02687
02688
         static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
02689
02690
          /* Check dimensions... */
          if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
02691
02692
            ERRMSG("Grid dimensions too large!");
02693
02694
          /\star Check quantity index for mass... \star/
02695
         if (ctl->qnt_m < 0)
02696
            ERRMSG("Need quantity mass!");
02697
02698
          /* Set time interval for output... */
          t0 = t - 0.5 * ctl->dt_mod;
02700
          t1 = t + 0.5 * ctl -> dt_mod;
02701
         /* Set grid box size... */
dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
02702
02703
02704
02705
         dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
02706
02707
          /* Initialize grid... */
         /* Initialize grid... */
for (ix = 0; ix < ctl->grid_nx; ix++)
    for (iy = 0; iy < ctl->grid_ny; iy++)
    for (iz = 0; iz < ctl->grid_nz; iz++)
02708
02709
02710
02711
                 mass[ix][iy][iz] = 0;
02712
02713
          /* Average data... */
         for (ip = 0; ip < atm->np; ip++)
  if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {</pre>
02714
02715
02716
02717
               /* 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);
02718
02719
02720
02721
               /* Check indices... */
if (ix < 0 || ix >= ctl->grid_nx ||
02722
02723
02724
                    iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
02725
02726
               /* Add mass... */
mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02727
02728
02729
02730
```

```
/* Check if gnuplot output is requested... */
02732
        if (ctl->grid_gpfile[0] != '-') {
02733
02734
           /\star Write info... \star/
02735
           printf("Plot grid data: %s.png\n", filename);
02736
02737
           /* Create gnuplot pipe... */
           if (!(out = popen("gnuplot", "w")))
02738
02739
             ERRMSG("Cannot create pipe to gnuplot!");
02740
           02741
02742
02743
02744
           /* Set time string... */
           jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02745
02746
02747
                    year, mon, day, hour, min);
02748
           /* Dump gnuplot file to pipe... */
02750
           if (!(in = fopen(ctl->grid_gpfile, "r")))
02751
             ERRMSG("Cannot open file!");
           while (fgets(line, LEN, in))
fprintf(out, "%s", line);
02752
02753
02754
           fclose(in);
02755
02756
02757
02758
02759
           /* Write info... */
          printf("Write grid data: %s\n", filename);
02760
02761
02762
           /* Create file... */
02763
           if (!(out = fopen(filename, "w")))
02764
             ERRMSG("Cannot create file!");
02765
02766
02767
         /* Write header... */
02768
        fprintf(out,
02769
                  "# $1 = time [s] \n"
                  "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n"
02770
02771
                  "# $4 = latitude [deg]\n"
02772
                 "# $5 = surface area [km^2]\n"
"# $6 = layer width [km]\n"
02773
02774
02775
                  "# $7 = temperature [K] \n"
02776
                  "# $8 = column density [kg/m^2] n"
02777
                  "# $9 = volume mixing ratio [1]\n\n");
02778
02779
        /* Write data... */
        for (ix = 0; ix < ctl->grid_nx; ix++) {
02780
         if
              (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
02782
             fprintf(out, "\n");
           for (iy = 0; iy < ctl->grid_ny; iy++) {
02783
            if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
    fprintf(out, "\n");
for (iz = 0; iz < ctl->grid_nz; iz++)
02784
02785
02786
02787
               if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
02788
                 /* Set coordinates... */
02789
                 z = ctl->grid_z0 + dz * (iz + 0.5);
lon = ctl->grid_lon0 + dlon * (ix + 0.5);
lat = ctl->grid_lat0 + dlat * (iy + 0.5);
02790
02791
02792
02794
                  /\star Get pressure and temperature... \star/
02795
                  press = P(z);
02796
                  intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
02797
                                   NULL, &temp, NULL, NULL, NULL, NULL, NULL, NULL);
02798
02799
                  /* Calculate surface area... */
                 * cos(lat * M_PI / 180.)
02801
02802
                 /* Calculate column density... */
cd = mass[ix][iy][iz] / (le6 * area);
02803
02804
02805
02806
                  /* Calculate volume mixing ratio... */
                  rho_air = 100. * press / (RA * temp);
vmr = MA / ctl->molmass * mass[ix][iy][iz]
02807
02808
02809
                    / (rho_air * 1e6 * area * 1e3 * dz);
02810
                 02811
02812
02813
02814
               }
02815
          }
02816
02817
```

```
02818  /* Close file... */
02819  fclose(out);
02820 }
```



5.21.2.39 void write\_prof ( const char \* filename, ctl t \* ctl, met t \* met0, met t \* met1, atm t \* atm, double t )

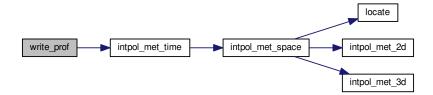
Write profile data.

Definition at line 2824 of file libtrac.c.

```
02830
                    {
02831
02832
        static FILE *in, *out;
02834
        static char line[LEN];
02835
        \verb|static double mass[GX][GY][GZ]|, obsmean[GX][GY]|, rt, rz, rlon, rlat, robs|,
02836
02837
          t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, vmr, h2o,
02838
02839
02840
        static int obscount[GX][GY], ip, ix, iy, iz, okay;
02841
02842
        /* Init... */
02843
        if (t == ctl->t_start) {
02844
02845
           /\star Check quantity index for mass... \star/
02846
          if (ctl->qnt_m < 0)
02847
            ERRMSG("Need quantity mass!");
02848
02849
          /* Check dimensions... */
if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
02850
02851
             ERRMSG("Grid dimensions too large!");
02852
02853
           /\star Open observation data file... \star/
02854
          printf("Read profile observation data: %s\n", ctl->prof_obsfile);
02855
           if (!(in = fopen(ctl->prof_obsfile, "r")))
            ERRMSG("Cannot open file!");
02856
02857
           /* Create new output file... */
          printf("Write profile data: %s\n", filename);
if (!(out = fopen(filename, "w")))
02859
02860
            ERRMSG("Cannot create file!");
02861
02862
02863
           /* Write header... */
02864
           fprintf(out,
02865
                           = time [s]\n"
02866
                   "# $2
                          = altitude [km] \n"
                    "# $3 = longitude [deg] \n"
02867
                    "# $4 = latitude [deg]\n"
02868
                    "# $5 = pressure [hPa]\n"
02869
02870
                   "# $6
                          = temperature [K]\n"
02871
                   "# $7 = volume mixing ratio [1]\n"
02872
                   "# $8 = H2O volume mixing ratio [1]\n"
                   "# $9 = O3 volume mixing ratio [1]\n"
"# $10 = mean BT index [K]\n");
02873
02874
02875
02876
           /* Set grid box size... */
          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;
02879
02880
02881
02882
         /* Set time interval... */
t0 = t - 0.5 * ctl->dt_mod;
02883
          t1 = t + 0.5 * ctl->dt_mod;
02885
          /* Initialize... */
for (ix = 0; ix < ctl->prof_nx; ix++)
    for (iy = 0; iy < ctl->prof_ny; iy++) {
02886
02887
02888
              obsmean[ix][iy] = 0;
02889
02890
               obscount[ix][iy] = 0;
               for (iz = 0; iz < ctl->prof_nz; iz++)
02891
02892
                 mass[ix][iy][iz] = 0;
02893
02894
02895
          /* Read observation data... */
          while (fgets(line, LEN, in)) {
02896
02897
            /* Read data... */
if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02898
02899
02900
                 5)
02901
               continue;
02902
02903
            /* Check time... */
02904
            if (rt < t0)
02905
              continue;
            if (rt > t1)
02906
02907
              break:
02908
02909
            /* Calculate indices... */
            ix = (int) ((rlon - ctl->prof_lon0) / dlon);
iy = (int) ((rlat - ctl->prof_lat0) / dlat);
02910
02911
02912
            /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
02913
02914
02915
              continue;
02916
02917
             /* Get mean observation index... */
02918
            obsmean[ix][iy] += robs;
            obscount[ix][iy]++;
02919
02920
02921
02922
          /* Analyze model data... */
02923
          for (ip = 0; ip < atm->np; ip++) {
02924
            /* Check time... */    if (atm->time[ip] < t0 || atm->time[ip] > t1)
02925
02926
02927
              continue:
02928
02929
             /* Get indices... */
            ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
02930
02931
02932
02933
02934
            /* Check indices... */
02935
            if (ix < 0 || ix >= ctl->prof_nx ||
02936
                 iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
02937
               continue;
02938
            /* Get total mass in grid cell... */
mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02939
02940
02941
02942
          /* Extract profiles... */
02943
          for (ix = 0; ix < ctl->prof_nx; ix++)
  for (iy = 0; iy < ctl->prof_ny; iy++)
02944
02945
02946
              if (obscount[ix][iy] > 0) {
02947
02948
                  /* Check profile... */
02949
                 okay = 0;
                  for (iz = 0; iz < ctl->prof_nz; iz++)
02950
                   if (mass[ix][iy][iz] > 0)
02951
02952
                      okay = 1;
02953
                 if (!okay)
02954
                    continue;
02955
                 /* Write output... */
fprintf(out, "\n");
02956
02957
02958
                  /* Loop over altitudes... */
02960
                 for (iz = 0; iz < ctl->prof_nz; iz++) {
02961
02962
                    /* Set coordinates... */
                   z = ctl->prof_z0 + dz * (iz + 0.5);
lon = ctl->prof_lon0 + dlon * (ix + 0.5);
02963
02964
```

```
lat = ctl - prof_lat0 + dlat * (iy + 0.5);
02966
02967
               /* Get pressure and temperature... */
02968
               press = P(z);
               02969
02970
02971
02972
               /* Calculate surface area... */
              area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
 * cos(lat * M_PI / 180.);
02973
02974
02975
02976
               /* Calculate volume mixing ratio... */
               rho_air = 100. * press / (RA * temp);
vmr = MA / ctl->molmass * mass[ix][iy][iz]
02977
02978
02979
                / (rho_air * area * dz * 1e9);
02980
               02981
02982
02983
02984
02985
02986
           }
02987
       /* Close file... */
if (t == ctl->t_stop)
02988
02989
         fclose(out);
02991 }
```



5.21.2.40 void write\_station ( const char \* filename, ctl\_t \* ctl, atm\_t \* atm, double t )

Write station data.

Definition at line 2995 of file libtrac.c.

```
02999
                    {
03000
03001
        static FILE *out:
03002
        static double rmax2, t0, t1, x0[3], x1[3];
03003
03004
03005
        static int ip, iq;
03006
03007
        /* Init... */
03008
        if (t == ctl->t_start) {
03009
03010
           /* Write info... */
03011
          printf("Write station data: %s\n", filename);
03012
          /* Create new file... */
if (!(out = fopen(filename, "w")))
03013
03014
             ERRMSG("Cannot create file!");
03015
03016
03017
           /* Write header... */
           fprintf(out,
03018
                    "# $1 = time [s] \n"
03019
                   "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03020
03021
03022
           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");
03024
03025
03026
03027
           /\star Set geolocation and search radius... \star/
03028
           geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
           rmax2 = gsl_pow_2(ctl->stat_r);
03029
03030
03031
        /* Set time interval for output... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
03032
03033
03034
03035
03036
         /* Loop over air parcels... */
03037
         for (ip = 0; ip < atm->np; ip++) {
03038
03039
           /\star Check time... \star/
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
03040
03041
             continue;
03042
03043
           /* Check station flag... */
03044
           if (ctl->qnt_stat >= 0)
            if (atm->q[ctl->qnt_stat][ip])
03045
03046
               continue;
03047
03048
           /* Get Cartesian coordinates... */
03049
           geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
03050
03051
           /\star Check horizontal distance... \star/
           if (DIST2(x0, x1) > rmax2)
03052
03053
             continue:
03054
03055
           /* Set station flag... */
03056
           if (ctl->qnt\_stat >= 0)
03057
             atm->q[ctl->qnt_stat][ip] = 1;
03058
           /* Write data... */
fprintf(out, "%.2f %g %g %g",
03059
03060
03061
                   atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
           for (iq = 0; iq < ctl->nq; iq++) {
   fprintf(out, " ");
03062
03063
             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
03064
03065
03066
           fprintf(out, "\n");
03067
03068
03069
         /* Close file... */
        if (t == ctl->t_stop)
03070
03071
           fclose(out);
03072 }
```

Here is the call graph for this function:



### 5.22 libtrac.h

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

5.22 libtrac.h 177

```
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) 2013-2018 Forschungszentrum Juelich GmbH
00018 */
00019
00035 #include <ctype.h>
00036 #include <gsl/gsl_math.h>
00037 #include <gsl/gsl_randist.h>
00038 #include <gsl/gsl_rng.h>
00039 #include <gsl/gsl_sort.h>
00040 #include <gsl/gsl_spline.h>
00041 #include <gsl/gsl_statistics.h>
00042 #include <math.h>
00043 #include <netcdf.h>
00044 #include <omp.h>
00045 #include <stdio.h>
00046 #include <stdlib.h>
00047 #include <string.h>
00048 #include <time.h>
00049 #include <sys/time.h>
00050
00051 /* -----
00052
        Constants...
00053
00054
00056 #define G0 9.80665
00057
00059 #define H0 7.0
00060
00062 #define KB 1.3806504e-23
00063
00065 #define MA 28.9644
00066
00068 #define P0 1013.25
00069
00071 #define RA 287.058
00072
00074 #define RI 8.3144598
00075
00077 #define RE 6367.421
00078
00079 /* -
08000
        Dimensions...
00081
00082
00084 #define LEN 5000
00085
00087 #define NP 10000000
00088
00090 #define NQ 12
00091
00093 #define EP 111
00094
00096 #define EX 1201
00097
00099 #define EY 601
00100
00102 #define GX 720
00103
00105 #define GY 360
00106
00108 #define GZ 100
00109
00111 #define NENS 2000
00112
00114 #define NTHREADS 512
00115
00116 /* -----
00117
         Macros...
00118
00119
00121 #define ALLOC(ptr, type, n)
00122
       if((ptr=calloc((size_t)(n), sizeof(type))) ==NULL)
00123
         ERRMSG("Out of memory!");
00124
00126 #define DIST(a, b) sqrt(DIST2(a, b))
00127
00129 #define DIST2(a, b)
00130
       ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))
00131
00133 #define DOTP(a, b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
00134
00136 #define ERRMSG(msg) {
```

```
printf("\nError (%s, %s, 1%d): %s\n\n",
00138
                    FILE__,
                             __func__, __LINE__, msg);
00139
         exit(EXIT_FAILURE);
       }
00140
00141
00143 #define FREAD(ptr, type, size, out) {
00144 if(fread(ptr, sizeof(type), size, out)!=size)
00145
            ERRMSG("Error while reading!");
00146
00147
00149 #define FWRITE(ptr, type, size, out) {
00150    if(fwrite(ptr, sizeof(type), size, out)!=size)
            ERRMSG("Error while writing!");
00151
00152
00153
00157
00159 #define NC(cmd) {
00160 if((cmd)!=NC_NOERR)
00161
           ERRMSG(nc_strerror(cmd));
00162
00163
00165 #define NORM(a) sgrt(DOTP(a, a))
00166
00168 #define PRINT(format, var)
      printf("Print (%s, %s, 1%d): %s= "format"\n",
00169
00170
              __FILE__, __func__, __LINE__, #var, var);
00171
00173 #define P(z) (P0*exp(-(z)/H0))
00174
00176 #define THETA(p, t) ((t)*pow(1000./(p), 0.286))
00177
00179 #define TOK(line, tok, format, var) {
00180         if(((tok)=strtok((line), " \t"))) {
00181             if(sscanf(tok, format, &(var))!=1) continue;
          } else ERRMSG("Error while reading!");
00182
00183
00184
00186 #define Z(p) (H0*log(P0/(p)))
00187
00188 /* -----
00189
        Timers...
00190
00191
00193 #define START_TIMER(id) timer(#id, id, 1)
00194
00196 #define STOP_TIMER(id) timer(#id, id, 2)
00197
00199 #define PRINT_TIMER(id) timer(#id, id, 3)
00200
00202 #define NTIMER 12
00203
00205 #define TIMER_TOTAL 0
00206
00208 #define TIMER_INIT 1
00211 #define TIMER_INPUT 2
00212
00214 #define TIMER OUTPUT 3
00215
00217 #define TIMER_ADVECT 4
00218
00220 #define TIMER_DECAY 5
00221
00223 #define TIMER_DIFFMESO 6
00224
00226 #define TIMER DIFFTURB 7
00227
00229 #define TIMER_ISOSURF 8
00230
00232 #define TIMER_METEO 9
00233
00235 #define TIMER POSITION 10
00236
00238 #define TIMER_SEDI 11
00239
00240 /* -----
        Structs...
00241
00242
00243
00245 typedef struct {
00246
00248
        int nq;
00249
00251
        char qnt_name[NQ][LEN];
00252
```

5.22 libtrac.h 179

```
00254
        char qnt_unit[NQ][LEN];
00255
00257
        char qnt_format[NQ][LEN];
00258
00260
        int qnt_ens;
00261
        int qnt_m;
00264
00266
        int qnt_rho;
00267
00269
        int qnt_r;
00270
00272
        int qnt_ps;
00273
00275
        int qnt_pt;
00276
00278
        int qnt_z;
00279
00281
        int qnt_p;
00282
00284
        int qnt_t;
00285
00287
        int qnt_u;
00288
00290
        int qnt_v;
00291
00293
        int qnt_w;
00294
00296
        int qnt_h2o;
00297
00299
        int qnt_o3;
00300
00302
        int qnt_theta;
00303
00305
        int qnt_vh;
00306
00308
        int qnt_vz;
00309
00311
        int qnt_pv;
00312
00314
        int qnt_tice;
00315
00317
        int qnt_tsts;
00318
00320
        int qnt_tnat;
00321
00323
        int qnt_stat;
00324
00326
        int direction;
00327
00329
        double t_start;
00330
00332
        double t_stop;
00333
00335
        double dt_mod;
00336
        double dt_met;
00339
00341
        int met_dx;
00342
        int met_dy;
00344
00345
00347
        int met_dp;
00348
00350
        int met_sx;
00351
00353
        int met_sy;
00354
00356
        int met_sp;
00357
00359
        int met_np;
00360
00362
        double met_p[EP];
00363
00366
        int met_tropo;
00367
00369
        char met_geopot[LEN];
00370
00372
        char met_stage[LEN];
00373
00376
        int isosurf;
00377
00379
        char balloon[LEN];
00380
00382
        double turb_dx_trop;
00383
00385
        double turb dx strat;
```

```
00386
00388
        double turb_dz_trop;
00389
00391
        double turb_dz_strat;
00392
00394
        double turb mesox:
00395
00397
        double turb_mesoz;
00398
00400
        double molmass;
00401
00403
        double tdec_trop;
00404
00406
        double tdec_strat;
00407
00409
        double psc_h2o;
00410
00412
        double psc_hno3;
00413
00415
        char atm_basename[LEN];
00416
00418
        char atm_gpfile[LEN];
00419
00421
        double atm dt out;
00422
00424
        int atm_filter;
00425
00427
        int atm_type;
00428
00430
        char csi_basename[LEN];
00431
00433
        double csi_dt_out;
00434
00436
        char csi_obsfile[LEN];
00437
        double csi_obsmin;
00439
00440
        double csi_modmin;
00443
00445
        int csi_nz;
00446
        double csi_z0;
00448
00449
00451
        double csi_z1;
00452
00454
        int csi_nx;
00455
00457
        double csi_lon0;
00458
00460
        double csi_lon1;
00461
00463
        int csi_ny;
00464
00466
        double csi_lat0;
00467
00469
        double csi lat1;
00470
00472
        char grid_basename[LEN];
00473
00475
        char grid_gpfile[LEN];
00476
00478
        double grid_dt_out;
00479
00481
        int grid_sparse;
00482
00484
        int grid_nz;
00485
00487
        double grid_z0;
00488
00490
        double grid_z1;
00491
00493
        int grid_nx;
00494
        double grid_lon0;
00496
00497
00499
        double grid_lon1;
00500
00502
00503
        int grid_ny;
00505
        double grid_lat0;
00506
00508
        double grid_lat1;
00509
00511
        char prof_basename[LEN];
00512
00514
        char prof_obsfile[LEN];
00515
```

5.22 libtrac.h 181

```
00517
        int prof_nz;
00518
00520
        double prof_z0;
00521
        double prof_z1;
00523
00524
        int prof_nx;
00527
00529
        double prof_lon0;
00530
00532
        double prof_lon1;
00533
        int prof_ny;
00536
00538
        double prof_lat0;
00539
00541
        double prof_lat1;
00542
00544
        char ens_basename[LEN];
00545
00547
        char stat_basename[LEN];
00548
        double stat_lon;
00551
00553
        double stat_lat;
00554
00556
        double stat_r;
00557
00558 } ctl_t;
00559
00561 typedef struct {
00562
00564
        int np;
00565
00567
        double time[NP];
00568
00570
        double p[NP];
00571
00573
        double lon[NP];
00574
00576
        double lat[NP];
00577
00579
        double q[NQ][NP];
00580
        float up[NP];
00583
00585
        float vp[NP];
00586
00588
        float wp[NP];
00589
00590 } atm_t;
00591
00593 typedef struct {
00594
00596
        double time;
00597
        int nx;
00600
00602
        int ny;
00603
00605
        int np;
00606
00608
        double lon[EX];
00609
00611
        double lat[EY];
00612
00614
        double p[EP];
00615
00617
        double ps[EX][EY];
00618
00620
        double pt[EX][EY];
00621
00623
        float z[EX][EY][EP];
00624
00626
        float t[EX][EY][EP];
00627
00629
        float u[EX][EY][EP];
00630
        float v[EX][EY][EP];
00632
00633
00635
        float w[EX][EY][EP];
00636
00638
        float pv[EX][EY][EP];
00639
00641
        float h2o[EX][EY][EP];
00642
        float o3[EX][EY][EP];
00644
```

```
00645
00647
        float pl[EX][EY][EP];
00648
00649 } met_t;
00650
00651 /* -
00652
        Functions...
00653
00654
00656 void cart2geo(
        double *x,
00657
00658
        double *z.
        double *lon,
00659
00660
        double *lat);
00661
00663 double clim_hno3(
       double t, double lat,
00664
00665
        double p);
00666
00667
00669 double clim_tropo(
00670
        double t,
00671
        double lat);
00672
00674 void day2doy(
00675
       int year,
00676
        int mon,
00677
       int day,
       int *doy);
00678
00679
00681 void doy2day(
00682
        int year,
00683
        int doy,
00684
        int *mon,
00685
        int *day);
00686
00688 double deg2dx(
00689
      double dlon,
00690
        double lat);
00691
00693 double deg2dy(
00694
       double dlat);
00695
00697 double dp2dz(
00698
      double dp,
00699
        double p);
00700
00702 double dx2deg(
00703
       double dx, double lat);
00704
00705
00707 double dy2deg(
00708
        double dy);
00709
00711 double dz2dp(
00712
        double dz,
00713
        double p);
00714
00716 void geo2cart(
00717
        double z,
00718
        double lon,
00719
        double lat,
00720
       double *x);
00721
00723 void get_met(
00724 ctl_t * ctl,
00725 char *metbase,
00726
        double t,
       met_t * met0,
met_t * met1);
00727
00728
00729
00731 void get_met_help(
00732
        double t,
00733
        int direct,
00734
        char *metbase,
00735
        double dt_met,
00736
        char *filename);
00737
00739 void intpol_met_2d(
        double array[EX][EY],
00740
00741
        int ix,
00742
        int iy,
00743
        double wx,
00744
        double wy,
00745
        double *var);
00746
00748 void intpol_met_3d(
```

5.22 libtrac.h 183

```
float array[EX][EY][EP],
00750
        int ip,
00751
        int ix,
00752
        int iy,
00753
        double wp,
00754
        double wx.
00755
        double wy,
00756
        double *var);
00757
00759 void intpol_met_space(
00760
        met_t * met,
        double p, double lon,
00761
00762
00763
        double lat,
00764
        double *ps,
00765
        double *pt,
00766
        double *z,
00767
        double *t,
00768
        double *u,
00769
        double *v,
00770
        double *w,
00771
        double *pv,
00772
        double *h2o,
00773
        double *o3);
00774
00776 void intpol_met_time(
       met_t * met0,
met_t * met1,
00777
00778
00779
        double ts,
00780
        double p,
00781
        double lon,
00782
        double lat,
00783
        double *ps,
00784
        double *pt,
00785
        double *z,
00786
        double *t,
00787
        double *u,
00788
        double *v,
00789
        double *w,
00790
        double *pv,
        double *h2o,
00791
00792
        double *o3);
00793
00795 void jsec2time(
00796
       double jsec,
00797
        int *year,
00798
        int *mon,
00799
        int *day,
00800
        int *hour,
00801
        int *min.
00802
        int *sec,
00803
        double *remain);
00804
00806 int locate(
00807
        double *xx,
80800
        int n,
        double x);
00810
00812 void read_atm(
00813
        const char *filename,
       ctl_t * ctl,
atm_t * atm);
00814
00815
00816
00818 void read_ctl(
00819
       const char *filename,
00820
       int argc,
00821
       char *argv[],
ctl_t * ctl);
00822
00823
00825 void read_met(
00826 ctl_t * ctl,
00827
        char *filename,
00828
       met_t * met);
00829
00831 void read_met_extrapolate(
00832 met_t * met);
00833
00835 void read_met_geopot(
       ctl_t * ctl,
met_t * met);
00836
00837
00838
00840 void read_met_help(
00841
       int ncid,
00842
        char *varname,
00843
        char *varname2,
       met_t * met,
float dest[EX][EY][EP],
00844
00845
```

```
00846
       float scl);
00847
00849 void read_met_ml2pl(
       ctl_t * ctl,
met_t * met,
00850
00851
00852
        float var[EX][EY][EP]);
00855 void read_met_periodic(
00856 met_t * met);
00857
00859 void read_met_pv(
00860 met t * met);
00861
00863 void read_met_sample(
00864 ctl_t * ctl,
        met_t * met);
00865
00866
00868 void read_met_tropo(
       ctl_t * ctl,
met_t * met);
00869
00870
00871
00873 double scan_ctl(
00874
        const char *filename,
00875
        int argc,
        char *argv[],
const char *varname,
00876
00877
00878
        int arridx,
00879
        const char *defvalue,
00880
       char *value);
00881
00883 void time2jsec(
00884
        int year,
00885
        int mon,
00886
        int day,
00887
        int hour,
00888
        int min,
00889
        int sec,
00890
        double remain,
00891
        double *jsec);
00892
00894 void timer(
00895
        const char *name,
00896
        int id.
00897
        int mode);
00898
00900 void write_atm(
00901 const char *filename,
00902
        ctl_t * ctl,
atm_t * atm,
00903
00904
        double t);
00905
00907 void write_csi(
00908
        const char *filename,
        ctl_t * ctl,
atm_t * atm,
00909
00910
00911
        double t);
00912
00914 void write_ens(
00915 const char *filename,
        ctl_t * ctl,
atm_t * atm,
00916
00917
00918
       double t);
00919
00921 void write_grid(
00922
        const char *filename,
        ctl_t * ctl,
met_t * met0,
00923
00924
       met_t * met1,
atm_t * atm,
00925
00926
00927
        double t);
00928
00930 void write_prof(
00931
        const char *filename,
00932
        ctl_t * ctl,
met_t * met0,
00933
00934
        met_t * met1,
00935
        atm_t * atm,
00936
        double t);
00937
00939 void write station(
00940
       const char *filename,
        ctl_t * ctl,
atm_t * atm,
00941
00942
00943
        double t);
```

## 5.23 match.c File Reference

Calculate deviations between two trajectories.

### **Functions**

• int main (int argc, char \*argv[])

## 5.23.1 Detailed Description

Calculate deviations between two trajectories.

Definition in file match.c.

## 5.23.2 Function Documentation

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

Definition at line 27 of file match.c.

```
00029
                          {
00030
00031
         ctl_t ctl;
00032
00033
        atm_t *atm1, *atm2, *atm3;
00034
00035
        FILE *out;
00036
00037
        char filename[LEN]:
00038
00039
         double filter_dt, x1[3], x2[3], dh, dq[NQ], dv, 1h = 0, 1t = 0, 1v = 0;
00040
00041
        int filter, ip1, ip2, iq, n;
00042
00043
         /* Allocate... */
        ALLOC(atm1, atm_t, 1);
ALLOC(atm2, atm_t, 1);
00044
00045
00046
        ALLOC(atm3, atm_t, 1);
00047
        /* Check arguments... */
if (argc < 5)</pre>
00048
00049
00050
           ERRMSG("Give parameters: <ctl> <atm_test> <atm_ref> <outfile>");
00051
00052
         /* Read control parameters... */
00053
         read_ctl(argv[1], argc, argv, &ctl);
        filter = (int) scan_ctl(argv[1], argc, argv, "FILTER", -1, "0", NULL);
filter_dt = scan_ctl(argv[1], argc, argv, "FILTER_DT", -1, "0", NULL);
00054
00055
00056
00057
         /* Read atmospheric data...
00058
        read_atm(argv[2], &ctl, atm1);
00059
         read_atm(argv[3], &ctl, atm2);
00060
00061
        /* Write info... */
        printf("Write transport deviations: %s\n", argv[4]);
00062
00063
00064
        /* Create output file...
        if (!(out = fopen(argv[4], "w")))
    ERRMSG("Cannot create file!");
00065
00066
00067
00068
        /* Write header... */
00069
        fprintf(out,
00070
                   "# $1 = time [s] \n"
                  "# \$2 = altitude [km]\n"
"# \$3 = longitude [deg]\n" "# \$4 = latitude [deg]\n");
00071
00072
        for (iq = 0; iq < ctl.nq; iq++) fprintf(out, "# \$%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00073
00074
00075
                     ctl.qnt_unit[iq]);
00076
        fprintf(out,
```

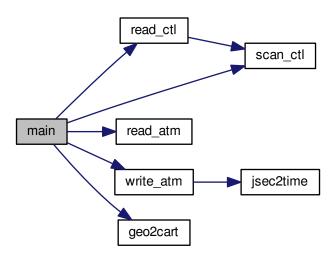
```
"# \$%d = trajectory time [s]\n"
00078
                   "# \$%d = vertical length of trajectory [km]\n"
                   "# \$%d = horizontal length of trajectory [km]\n"
00079
                   "# \$%d = vertical deviation [km]\n"
00080
                   "# \$%d = horizontal deviation [km]\n",
00081
00082
                   5 + ctl.nq, 6 + ctl.nq, 7 + ctl.nq, 8 + ctl.nq, 9 + ctl.nq);
         for (iq = 0; iq < ctl.nq; iq+)
fprintf(out, "# $%d = %s deviation [%s]\n", ctl.nq + iq + 10,
00084
         ctl.qnt_name[iq], ctl.qnt_unit[iq]);
fprintf(out, "\n");
00085
00086
00087
00088
         /\star Filtering of reference time series... \star/
00089
         if (filter) {
00090
00091
            /* Copy data... */
00092
           memcpy(atm3, atm2, sizeof(atm_t));
00093
00094
           /* Loop over data points... */
for (ip1 = 0; ip1 < atm2->np; ip1++) {
00095
00096
              n = 0;
00097
              atm2->p[ip1] = 0;
00098
              for (iq = 0; iq < ctl.nq; iq++)</pre>
              atm2->q[iq][ip1] = 0;
for (ip2 = 0; ip2 < atm2->np; ip2++)
00099
00100
00101
                 if (fabs(atm2->time[ip1] - atm2->time[ip2]) < filter_dt) {</pre>
                  atm2->p[ip1] += atm3->p[ip2];
00102
00103
                   for (iq = 0; iq < ctl.nq; iq++)</pre>
00104
                     atm2->q[iq][ip1] += atm3->q[iq][ip2];
00105
                   n++;
00106
                }
              atm2->p[ip1] /= n;
for (iq = 0; iq < ctl.nq; iq++)
  atm2->q[iq][ip1] /= n;
00107
00108
00109
00110
00111
           /* Write filtered data... */
sprintf(filename, "%s.filt", argv[3]);
00112
00113
           write_atm(filename, &ctl, atm2, 0);
00114
00115
00116
00117
          /* Loop over air parcels (reference data)... */
         for (ip2 = 0; ip2 < atm2->np; ip2++) {
00118
00119
00120
            /* Get trajectory length... */
           if (ip2 > 0) {
00121
00122
              geo2cart(0, atm2->lon[ip2 - 1], atm2->lat[ip2 - 1], x1);
00123
              geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00124
              lh += DIST(x1, x2);
              lv += fabs(Z(atm2->p[ip2 - 1]) - Z(atm2->p[ip2]));
lt = fabs(atm2->time[ip2] - atm2->time[0]);
00125
00126
00127
00128
00129
            /* Init... */
00130
           n = 0;

dh = 0;
00131
            dv = 0;
00132
            for (iq = 0; iq < ctl.nq; iq++)</pre>
             dq[iq] = 0;
00134
00135
            geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00136
00137
            /* Find corresponding time step (test data)... */
           for (ip1 = 0; ip1 < atml->np; ip1++)
  if (fabs(atml->time[ip1] - atm2->time[ip2])
  < (filter ? filter_dt : 0.1)) {</pre>
00138
00139
00140
00141
00142
                 /* Calculate deviations... */
00143
                 geo2cart(0, atm1->lon[ip1], atm1->lat[ip1], x1);
                 dh += DIST(x1, x2);
dv += Z(atm1->p[ip1]) - Z(atm2->p[ip2]);
00144
00145
                 for (iq = 0; iq < ctl.nq; iq++)</pre>
00146
00147
                   dq[iq] += atm1->q[iq][ip1] - atm2->q[iq][ip2];
00148
                n++;
00149
              }
00150
00151
            /* Write output... */
            if (n > 0) {
00152
00153
              fprintf(out, "%.2f %.4f %.4f %.4f",
00154
                      atm2->time[ip2], Z(atm2->p[ip2]),
00155
                        atm2->lon[ip2], atm2->lat[ip2]);
              for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl.qnt_format[iq], atm2->q[iq][ip2]);
00156
00157
00158
00159
              fprintf(out, " %.2f %g %g %g %g", lt, lv, lh, dv / n, dh / n);
00160
              for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl.qnt_format[iq], dq[iq] / n);</pre>
00161
00162
00163
```

5.24 match.c 187

```
00164
00165
            fprintf(out, "\n");
00166
       }
00167
00168
        /* Close file... */
00169
00170
        fclose(out);
00171
00172
        /* Free... */
00173
        free(atm1);
00174
        free (atm2);
00175
       free(atm3);
00176
00177
        return EXIT_SUCCESS;
00178 }
```

Here is the call graph for this function:



## 5.24 match.c

```
00001 /*
00002
         This file is part of MPTRAC.
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
         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 \frac{http://www.gnu.org/licenses}{}.
00016
00017
         Copyright (C) 2013-2018 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028
         int argc,
00029
         char *argv[]) {
00030
00031
         ctl_t ctl;
```

```
00033
        atm_t *atm1, *atm2, *atm3;
00034
00035
        FILE *Out:
00036
00037
        char filename[LEN]:
00039
         double filter_dt, x1[3], x2[3], dh, dq[NQ], dv, 1h = 0, 1t = 0, 1v = 0;
00040
00041
         int filter, ip1, ip2, iq, n;
00042
         /* Allocate... */
00043
        ALLOC(atm1, atm_t, 1);
ALLOC(atm2, atm_t, 1);
00044
00045
00046
         ALLOC(atm3, atm_t, 1);
00047
00048
         /* Check arguments... */
00049
         if (argc < 5)
00050
           ERRMSG("Give parameters: <ctl> <atm_test> <atm_ref> <outfile>");
00051
00052
         /* Read control parameters... */
        read_ctl(argv[1], argc, argv, &ctl);
filter = (int) scan_ctl(argv[1], argc, argv, "FILTER", -1, "0", NULL);
filter_dt = scan_ctl(argv[1], argc, argv, "FILTER_DT", -1, "0", NULL);
00053
00054
00055
00056
00057
         /* Read atmospheric data... */
00058
         read_atm(argv[2], &ctl, atml);
00059
        read_atm(argv[3], &ctl, atm2);
00060
        /* Write info... */
00061
00062
        printf("Write transport deviations: %s\n", argv[4]);
00063
        /* Create output file... */
if (!(out = fopen(argv[4], "w")))
00064
00065
          ERRMSG("Cannot create file!");
00066
00067
00068
         /* Write header... */
00069
         fprintf(out,
00070
                  "# $1 = time [s] \n"
                  "# \$2 = altitude [km]\n"
"# \$3 = longitude [deg]\n" "# \$4 = latitude [deg]\n");
00071
00072
        for (iq = 0; iq < ctl.nq; iq++)
  fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],</pre>
00073
00074
00075
                    ctl.qnt_unit[iq]);
00076
         fprintf(out,
                  "# \$%d = trajectory time [s]\n"
00077
                  "# \$%d = vertical length of trajectory [km]\n"
00078
                  "# \$%d = horizontal length of trajectory [km]\n"
00079
                  "# \$%d = vertical deviation [km]\n"
08000
                  "# $%d = horizontal deviation [km]\n",
00081
00082
                  5 + ctl.nq, 6 + ctl.nq, 7 + ctl.nq, 8 + ctl.nq, 9 + ctl.nq);
         for (iq = 0; iq < ctl.nq; iq++)
  fprintf(out, "# $%d = %s deviation [%s]\n", ctl.nq + iq + 10,</pre>
00083
00084
        ctl.qnt_name[iq], ctl.qnt_unit[iq]);
fprintf(out, "\n");
00085
00086
00087
00088
         /* Filtering of reference time series... */
00089
         if (filter) {
00090
00091
           /* Copy data... */
           memcpy(atm3, atm2, sizeof(atm_t));
00092
00093
00094
           /* Loop over data points... */
00095
           for (ip1 = 0; ip1 < atm2->np; ip1++) {
00096
             n = 0;
00097
              atm2->p[ip1] = 0;
             for (iq = 0; iq < ctl.nq; iq++)</pre>
00098
00099
               atm2->q[iq][ip1] = 0;
              for (ip2 = 0; ip2 < atm2->np; ip2++)
00100
               if (fabs(atm2->time[ip1] - atm2->time[ip2]) < filter_dt) {</pre>
00101
00102
                  atm2->p[ip1] += atm3->p[ip2];
00103
                  for (iq = 0; iq < ctl.nq; iq++)</pre>
                    atm2->q[iq][ip1] += atm3->q[iq][ip2];
00104
00105
                  n++;
00106
00107
             atm2->p[ip1] /= n;
00108
             for (iq = 0; iq < ctl.nq; iq++)</pre>
00109
               atm2->q[iq][ip1] /= n;
00110
00111
           /* Write filtered data... */
sprintf(filename, "%s.filt", argv[3]);
00112
00113
00114
           write_atm(filename, &ctl, atm2, 0);
00115
00116
00117
         /* Loop over air parcels (reference data)... */
        for (ip2 = 0; ip2 < atm2->np; ip2++) {
00118
```

```
00119
00120
           /* Get trajectory length... */
00121
           if (ip2 > 0) {
            geo2cart(0, atm2->lon[ip2 - 1], atm2->lat[ip2 - 1], x1);
00122
00123
             geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00124
             1h += DIST(x1, x2);
00125
             lv += fabs(Z(atm2->p[ip2 - 1]) - Z(atm2->p[ip2]));
00126
             lt = fabs(atm2->time[ip2] - atm2->time[0]);
00127
00128
           /* Init... */
00129
00130
          n = 0:
00131
           dh = 0;
           dv = 0;
00132
          for (iq = 0; iq < ctl.nq; iq++)
  dq[iq] = 0;</pre>
00133
00134
00135
           geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00136
00137
           /\star Find corresponding time step (test data)... \star/
00138
          for (ip1 = 0; ip1 < atm1->np; ip1++)
00139
            if (fabs(atm1->time[ip1] - atm2->time[ip2])
00140
                  < (filter ? filter_dt : 0.1)) {
00141
               /* Calculate deviations... */
geo2cart(0, atm1->lon[ip1], atm1->lat[ip1], x1);
00142
00143
               dh += DIST(x1, x2);
00144
00145
               dv += Z(atm1->p[ip1]) - Z(atm2->p[ip2]);
00146
               for (iq = 0; iq < ctl.nq; iq++)</pre>
00147
                 dq[iq] += atm1->q[iq][ip1] - atm2->q[iq][ip2];
00148
               n++;
00149
00150
00151
           /* Write output... */
00152
00153
            fprintf(out, "%.2f %.4f %.4f %.4f",
                     atm2->time[ip2], Z(atm2->p[ip2]),
00154
                     atm2->lon[ip2], atm2->lat[ip2]);
00155
             for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");</pre>
00156
00158
               fprintf(out, ctl.qnt_format[iq], atm2->q[iq][ip2]);
00159
             fprintf(out, " %.2f %g %g %g %g", lt, lv, lh, dv / n, dh / n);
for (iq = 0; iq < ctl.nq; iq++) {
   fprintf(out, " ");</pre>
00160
00161
00162
               fprintf(out, ctl.qnt_format[iq], dq[iq] / n);
00163
00164
00165
             fprintf(out, "\n");
00166
        }
00167
00168
00169
         /* Close file... */
00170
        fclose(out);
00171
00172
        /* Free... */
00173
        free(atm1);
00174
        free(atm2);
00175
        free (atm3);
00176
00177
        return EXIT_SUCCESS;
00178 }
```

# 5.25 met map.c File Reference

Extract global map from meteorological data.

## **Functions**

• int main (int argc, char \*argv[])

# 5.25.1 Detailed Description

Extract global map from meteorological data.

Definition in file met\_map.c.

#### 5.25.2 Function Documentation

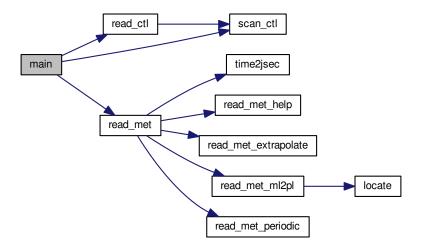
#### 5.25.2.1 int main ( int argc, char \* argv[] )

Definition at line 27 of file met map.c.

```
00029
00030
00031
        ctl_t ctl;
00032
00033
        met t *met;
00034
        FILE *out;
00036
00037
        \texttt{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];
00038
00039
00040
00041
        static int i, ip, ip2, ix, iy, np[EX][EY];
00042
         /* Allocate... */
00043
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_ctl(argv[1], argc, argv, &ctl);
z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00051
00052
00053
00054
         /* Loop over files... */
00055
        for (i = 3; i < argc; i++) {</pre>
00056
00057
           /\star Read meteorological data... \star/
00058
           read met(&ctl, argv[i], met);
00059
00060
           /\star Find nearest pressure level... \star/
00061
           for (ip2 = 0; ip2 < met->np; ip2++) {
00062
             dz = fabs(Z(met->p[ip2]) - z);
00063
             if (dz < dzmin) {</pre>
00064
               dzmin = dz;
00065
               ip = ip2;
00066
             }
00067
          }
00068
           /* Average data... */
for (ix = 0; ix < met->nx; ix++)
  for (iy = 0; iy < met->ny; iy++) {
00069
00070
00071
               intpol_met_space(met, met->pt[ix][iy], met->lon[ix], met->
00072
      lat[iy],
00073
                                  NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL,
00074
                                  NULL);
00075
               timem[ix][iy] += met->time;
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];
00080
               wm[ix][iy] += met->w[ix][iy][ip];
00081
               pvm[ix][iy] += met->pv[ix][iy][ip];
               h2om[ix][iy] += met->h2o[ix][iy][ip];
00082
00083
               o3m[ix][iy] += met->o3[ix][iy][ip];
               psm[ix][iy] += met->ps[ix][iy];
00084
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... */
        printf("Write meteorological data file: %s\n", argv[2]);
00093
        if (!(out = fopen(argv[2], "w")))
00094
          ERRMSG("Cannot create file!");
00095
00096
        /* Write header... */
00097
00098
        fprintf(out,
00099
                  "# $1
                         = time [s]\n"
                  "# $2 = altitude [km]\n"
00100
                 "# $3 = longitude [deg]\n"
00101
00102
                 "# $4 = latitude [deg]\n"
00103
                 "# $5 = pressure [hPa]\n"
```

```
"# $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 = H2O volume mixing ratio [1]\n");
00108
00109
                    fprintf(out,
                                            "# $11 = 03 volume mixing ratio [1]\n"
00110
00111
                                           "# $12 = geopotential height [km]\n
00112
                                           "# $13 = potential vorticity [PVU] \n"
                                           "# $14 = surface pressure [hPa] \n"
00113
                                           "# $15 = tropopause pressure [hPa]\n"
00114
                                            "# $16 = tropopause geopotential height [km] \n"
00115
00116
                                           "# $17 = tropopause temperature [K]\n");
00117
00118
                      /* Write data... */
                    00119
00120
00121
00124
00125
                                                            tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00126
00127
00128
00129
                                                            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]);
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],
vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00137
00138
                                                           hand in the problem of the prob
00139
00140
00141
00142
00143
00144
                     /* Close file... */
00145
00146
                    fclose(out):
00147
00148
                       /* Free... */
00149
                     free (met);
00150
00151
                     return EXIT_SUCCESS;
00152 }
```

Here is the call graph for this function:



# 5.26 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) 2013-2018 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],
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... */
        read_ctl(argv[1], argc, argv, &ctl);
z = scan_ctl(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;
00076
                zm[ix][iy] += met->z[ix][iy][ip];
00077
                tm[ix][iy] += met->t[ix][iy][ip];
                um[ix][iy] += met->u[ix][iy][ip];
00078
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];
                psm[ix][iy] += met->ps[ix][iy];
00084
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... */
        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
                 "# $3 = longitude [deg]\n"
00101
                "# $4 = latitude [deg]\n"
00102
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");
00108
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
                       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],
00140
00141
00142
                       ztm[ix][iy] / np[ix][iy], ttm[ix][iy] / np[ix][iy]);
00143
00144
        /* Close file... */
00146
        fclose(out);
00147
        /* Free... */
00148
00149
        free (met);
00150
00151
        return EXIT_SUCCESS;
00152 }
```

## 5.27 met\_prof.c File Reference

Extract vertical profile from meteorological data.

# **Functions**

• int main (int argc, char \*argv[])

#### 5.27.1 Detailed Description

Extract vertical profile from meteorological data.

Definition in file met\_prof.c.

#### 5.27.2 Function Documentation

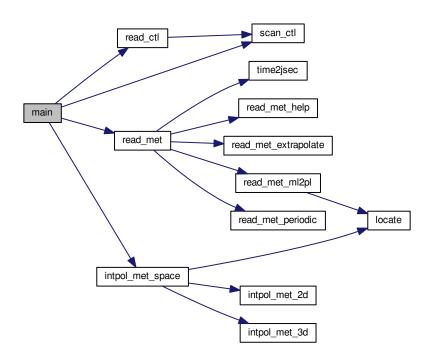
### 5.27.2.1 int main ( int argc, char \* argv[] )

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
             zg, zgm[NZ], zt, ztm[NZ], pv, pvm[NZ];
00051
00052
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... */
00063
          read_ctl(argv[1], argc, argv, &ctl);
          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, "LONO", -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, "LAT1", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "LAT1", -1, "0", NULL);
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++) {</pre>
00076
00077
             /* Read meteorological data... */
             read_met(&ctl, argv[i], met);
00078
00079
             /* Average... */
08000
             for (z = z0; z <= z1; z += dz) {</pre>
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
00103
                        o3m[iz] += o3;
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]);
```

```
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,
                  "# $10 = H20 volume mixing ratio [1]\n"
"# $11 = 03 volume mixing ratio [1]\n"
"# $12 = geopotential height [km]\n"
"# $13 = potential vorticity [PVU]\n"
00131
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:



# 5.28 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) 2013-2018 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 /* -----
           Main...
00035
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
            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
          if (argc < 4)
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);
00070
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);
00091
                if (gsl_finite(t) && gsl_finite(u)
                    && gsl_finite(v) && gsl_finite(w)) {
00092
00093
                 timem[iz] += met->time;
                 lonm[iz] += lon;
latm[iz] += lat;
00094
00095
                 zgm[iz] += zg;
00097
                 tm[iz] += t;
00098
                 um[iz] += u;
00099
                 vm[iz] += v;
00100
                 wm[iz] += w;
                 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
               }
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
00119
       /* Write header... */
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"
               "# $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
00154
       /* Free... */
00155
       free(met);
00156
00157
        return EXIT SUCCESS:
00158 }
```

## 5.29 met\_sample.c File Reference

Sample meteorological data at given geolocations.

#### **Functions**

• int main (int argc, char \*argv[])

#### 5.29.1 Detailed Description

Sample meteorological data at given geolocations.

Definition in file met sample.c.

#### 5.29.2 Function Documentation

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

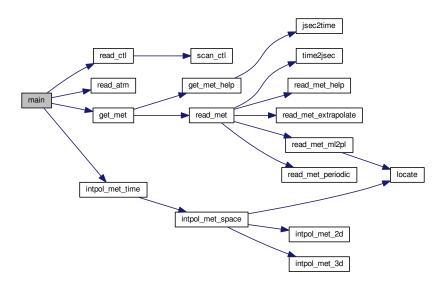
Definition at line 31 of file met\_sample.c.

```
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... */
        printf("Write meteorological data file: %s\n", argv[4]);
00065
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
                "# $6 = temperature [K]\n"
00076
                "# $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"
00083
                "# $13 = potential vorticity [PVU]\n"
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];
```

5.30 met sample.c 199

```
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
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
       /* 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:



#### 5.30 met sample.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
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) 2013-2018 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /*
00028
         Main...
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
                 "# $17 = tropopause temperature [K]\n\n");
00088
00089
00090
        /* Loop over air parcels... */
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);
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/
          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
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
       /* Close file... */
00128
00129
       fclose(out);
00130
00131
       /* Free... */
00132
       free (atm);
00133
       free (met0);
00134
       free (met1);
00135
00136
        return EXIT_SUCCESS;
00137 }
```

# 5.31 met\_zm.c File Reference

Extract zonal mean from meteorological data.

#### **Functions**

• int main (int argc, char \*argv[])

## 5.31.1 Detailed Description

Extract zonal mean from meteorological data.

Definition in file met\_zm.c.

#### 5.31.2 Function Documentation

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

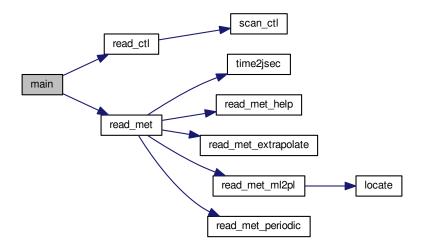
Definition at line 27 of file met zm.c.

```
00029
00030
00031
        ctl_t ctl;
00032
00033
        met t *met;
00034
00035
        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];
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];
00074
                 o3m[ip][iy] += met->o3[ix][iy][ip];
                 psm[ip][iy] += met->ps[ix][iy];
00075
                 ptm[ip][iy] += met->pt[ix][iy];
00076
                 ztm[ip][iy] += zt;
00077
00078
                 ttm[ip][iy] += tt;
00079
                 np[ip][iy]++;
00080
00081
        }
00082
        /* Create output file... */
00083
00084
        printf("Write meteorological data file: %s\n", argv[2]);
00085
        if (!(out = fopen(argv[2], "w")))
00086
          ERRMSG("Cannot create file!");
00087
00088
        /* Write header... */
00089
        fprintf(out,
                       = time [s] \n"
00090
                 "# $1
00091
                 "# $2
                        = altitude [km]\n"
00092
                 "# $3 = longitude [deg] \n"
                 "# $4 = latitude [deg]\n"
00093
                 "# $5 = pressure [hPa]\n"
00094
                 "# $6 = temperature [K]\n"
00095
                       = zonal wind [m/s]\n"
00096
00097
                 "# $8 = meridional wind [m/s]\n"
00098
                 "# $9 = vertical wind [hPa/s]\n"
                 "# $10 = H2O volume mixing ratio [1]\n");
00099
        fprintf(out,
    "# $11 = 03 volume mixing ratio [1]\n"
00100
00101
00102
                 "# $12 = geopotential height [km]\n'
00103
                 "# $13 = potential vorticity [PVU]\n"
```

5.32 met zm.c 203

```
"# $14 = surface pressure [hPa] \n"
00105
                    "# $15 = tropopause pressure [hPa]\n"
                     "# $16 = tropopause geopotential height [km]\n"
00106
                     "# $17 = tropopause temperature [K]\n");
00107
00108
          /* Write data... */
for (ip = 0; ip < met->np; ip++) {
  fprintf(out, "\n");
00109
00110
00111
             00112
00113
00114
                          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],
h2om[ip][iy] / np[ip][iy], o3m[ip][iy] / np[ip][iy],
00115
00116
00117
                          zm(ip)[iy] / np[ip)[iy], pvm[ip][iy] / np[ip][iy],
psm[ip][iy] / np[ip][iy], ptm[ip][iy] / np[ip][iy],
ztm[ip][iy] / np[ip][iy], ttm[ip][iy] / np[ip][iy]);
00118
00119
00120
00121
00122
          /* Close file... */
00123
00124
          fclose(out);
00125
00126
          /* Free... */
00127
         free (met);
00128
          return EXIT_SUCCESS;
00130 }
```

Here is the call graph for this function:



## 5.32 met zm.c

```
00001 /*
00002
        This file is part of MPTRAC.
00003
00004
        MPTRAC is free software: you can redistribute it and/or modify
00005
        it under the terms of the GNU General Public License as published by
        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
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) 2013-2018 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],
         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];
00042
        /* Allocate... */
00043
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
00053
        /* Loop over files... */
00054
        for (i = 3; i < argc; i++) {</pre>
00055
00056
           /* Read meteorological data... */
00057
          read_met(&ctl, argv[i], met);
00058
           /* Average data... */
00060
          for (ix = 0; ix < met->nx; ix++)
00061
           for (iy = 0; iy < met->ny; iy++)
00062
               for (ip = 0; ip < met->np; ip++) {
                intpol_met_space(met, met->pt[ix][iy], met->lon[ix], met->
00063
     lat[iv],
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];
00069
00070
                 vm[ip][iy] += met->v[ix][iy][ip];
                 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
                 ptm[ip][iy] += met->pt[ix][iy];
00077
                 ztm[ip][iy] += zt;
00078
                 ttm[ip][iy] += tt;
00079
                 np[ip][iy]++;
08000
              }
00081
        1
00082
00083
        /* Create output file... */
00084
        printf("Write meteorological data file: %s\n", argv[2]);
00085
        if (!(out = fopen(argv[2], "w")))
00086
          ERRMSG("Cannot create file!");
00087
00088
        /* Write header... */
00089
        fprintf(out,
00090
                 "# $1
                        = time [s]\n"
00091
                 "# $2 = altitude [km] \n"
                 "# $3 = longitude [deg] \n"
00092
                 "# $4 = latitude [deg]\n"
00093
                 "# $5 = pressure [hPa]\n"
00094
00095
                 "# $6 = temperature [K]\n"
00096
                 "# $7 = zonal wind [m/s]\n"
00097
                 "# $8 = meridional wind [m/s]\n"
00098
                 "# $9 = vertical wind [hPa/s]\n"
                 "# $10 = H2O volume mixing ratio [1]\n");
00099
00100
        fprintf(out,
                 "# $11 = 03 volume mixing ratio [1]\n"
00101
                 "# $12 = geopotential height [km]\n'
00102
00103
                 "# $13 = potential vorticity [PVU]\n"
                 "# $14 = surface pressure [hPa]\n"
00104
                 "# $15 = tropopause pressure [hPa] \n"
00105
                 "# $16 = tropopause geopotential height [km]\n"
00106
                 "# $17 = tropopause temperature [K]\n");
00107
```

```
00109
         /* Write data... */
         for (ip = 0; ip < met->np; ip++) {
  fprintf(out, "\n");
00110
00111
           00112
00113
00114
00115
                      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],
00116
                      h2om[ip][iy] / np[ip][iy], o3m[ip][iy] / np[ip][iy],
00117
                      zm[ip][iy] / np[ip][iy], pvm[ip][iy] / np[ip][iy],
psm[ip][iy] / np[ip][iy], ptm[ip][iy] / np[ip][iy],
ztm[ip][iy] / np[ip][iy], ttm[ip][iy] / np[ip][iy]);
00118
00119
00120
00121
00122
00123
        /* Close file... */
00124
        fclose(out);
00125
00126
        /* Free... */
00127
        free (met);
00128
00129
         return EXIT_SUCCESS;
00130 }
```

# 5.33 smago.c File Reference

Estimate horizontal diffusivity based on Smagorinsky theory.

## **Functions**

• int main (int argc, char \*argv[])

# 5.33.1 Detailed Description

Estimate horizontal diffusivity based on Smagorinsky theory.

Definition in file smago.c.

## 5.33.2 Function Documentation

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

Definition at line 8 of file smago.c.

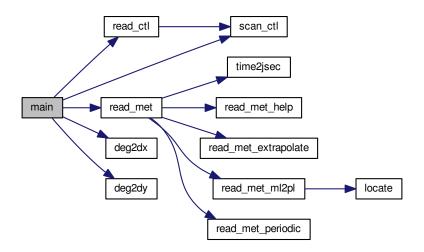
```
00010
00011
00012
        ctl_t ctl;
00013
00014
       met_t *met;
00015
00016
       FILE *out;
00017
00018
       static double dz, dzmin = 1e10, z, t, s, ls2, k[EX][EY], c = 0.15;
00019
00020
       static int ip, ip2, ix, iy;
00021
00022
       /* Allocate... */
00023
       ALLOC(met, met_t, 1);
00024
00025
        /* Check arguments... */
00026
       if (argc < 4)
00027
         ERRMSG("Give parameters: <ctl> <map.tab> <met>");
00028
00029
       /* Read control parameters... */
00030
       read_ctl(argv[1], argc, argv, &ctl);
```

```
z = scan_{ctl}(argv[1], argc, argv, "Z", -1, "", NULL);
00032
00033
         /* Read meteorological data... */
00034
        read_met(&ctl, argv[3], met);
00035
00036
         /* Find nearest pressure level... */
         for (ip2 = 0; ip2 < met->np; ip2++) {
00038
                fabs(Z(met->p[ip2]) - z);
          if (dz < dzmin) {
  dzmin = dz;</pre>
00039
00040
00041
             ip = ip2;
00042
           }
00043
        }
00044
         /* Write info... */
00045
00046
        printf("Analyze %g hPa...\n", met->p[ip]);
00047
00048
         /* Calculate horizontal diffusion coefficients... */
         for (ix = 1; ix < met->nx - 1; ix++)
for (iy = 1; iy < met->ny - 1; iy++) {
00050
00051
             t = 0.5 * ((met->u[ix + 1][iy][ip] - met->u[ix - 1][iy][ip])
00052
                          / (1000. *
00053
                              deg2dx (met->lon[ix + 1] - met->lon[ix - 1], met->
      lat[iy]))
00054
                          - (met->v[ix][iy + 1][ip] - met->v[ix][iy - 1][ip])
00055
             / (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])
00056
00057
                          / (1000. * deg2dy(met->lat[iy + 1] - met->lat[iy - 1]))
00058
                          + (met->v[ix + 1][iy][ip] - met->v[ix - 1][iy][ip])
00059
                          / (1000. *
                             deg2dx(met->lon[ix + 1] - met->lon[ix - 1],
00060
00061
                                     met->lat[iy])));
00062
              ls2 = gsl_pow_2(c * 500. * deg2dy(met->lat[iy + 1] - met->lat[iy - 1]));
00063
             if (fabs(met->lat[iy]) > 80)
              ls2 *= (90. - fabs(met->lat[iy])) / 10.; \\ k[ix][iy] = ls2 * sqrt(2.0 * (gsl_pow_2(t) + gsl_pow_2(s))); 
00064
00065
00066
00067
00068
         /* Create output file... */
00069
        printf("Write data file: %s\n", argv[2]);
00070
         if (!(out = fopen(argv[2], "w")))
          ERRMSG("Cannot create file!");
00071
00072
00073
         /* Write header... */
00074
        fprintf(out,
00075
                  "# $1 = longitude [deg] \n"
                  "# $2 = latitude [deg]\n"
"# $3 = zonal wind [m/s]\n"
00076
00077
                  "# $4 = meridional wind [m/s]\n"
00078
                  "# $5 = horizontal diffusivity [m^2/s]\n");
00079
00080
00081
         /* Write data... */
         for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
  for (ix = 0; ix < met->nx; ix++)
    if (met->lon[ix] >= 180)
      fprintf(out, "%g %g %g %g %g\n",
00082
00083
00084
00085
00087
                         met->lon[ix] - 360.0, met->lat[iy],
00088
                         met->u[ix][iy][ip], met->v[ix][iy][ip], k[ix][iy]);
00089
           for (ix = 0; ix < met->nx; ix++)
             00090
00091
00092
00093
                         met->u[ix][iy][ip], met->v[ix][iy][ip], k[ix][iy]);
00094
00095
        /* Close file... */
00096
00097
        fclose(out);
00098
00099
         /* Free... */
00100
        free (met);
00101
00102
        return EXIT_SUCCESS;
00103 }
```

5.34 smago.c 207

Here is the call graph for this function:



# 5.34 smago.c

```
00001
00006 #include "libtrac.h"
00008 int main(
00009
        int argc,
        char *argv[]) {
00010
00011
00012
        ctl t ctl:
00013
00014
        met_t *met;
00015
00016
        FILE *out;
00017
00018
        static double dz, dzmin = 1e10, z, t, s, 1s2, k[EX][EY], c = 0.15;
00019
00020
         static int ip, ip2, ix, iy;
00021
00022
         /* Allocate... */
00023
        ALLOC(met, met_t, 1);
00024
00025
         /* Check arguments... */
00026
         if (argc < 4)
00027
           ERRMSG("Give parameters: <ctl> <map.tab> <met>");
00028
00029
        /* Read control parameters... */
        read_ctl(argv[1], argc, argv, &ctl);
z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00030
00031
00032
00033
         /* Read meteorological data... */
00034
         read_met(&ctl, argv[3], met);
00035
00036
         /* Find nearest pressure level... */
         for (ip2 = 0; ip2 < met->np; ip2++) {
  dz = fabs(Z(met->p[ip2]) - z);
00037
00038
00039
           if (dz < dzmin) {
00040
             dzmin = dz;
00041
              ip = ip2;
00042
           }
00043
00044
00045
         /* Write info... */
00046
         printf("Analyze %g hPa...\n", met->p[ip]);
00047
         /* Calculate horizontal diffusion coefficients... */
for (ix = 1; ix < met->nx - 1; ix++)
  for (iy = 1; iy < met->ny - 1; iy++) {
00048
00049
00050
00051
             t = 0.5 * ((met -> u[ix + 1][iy][ip] - met -> u[ix - 1][iy][ip])
```

```
00052
                         / (1000. *
                             deg2dx(met->lon[ix + 1] - met->lon[ix - 1], met->
      lat[iy]))
00054
                         - (met->v[ix][iy + 1][ip] - met->v[ix][iy - 1][ip])
             / (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])

/ (1000. * deg2dy(met->lat[iy + 1] - met->lat[iy - 1]))
00055
00056
00058
                          + (met -> v[ix + 1][iy][ip] - met -> v[ix - 1][iy][ip])
                          / (1000. *
00059
00060
                             deg2dx(met->lon[ix + 1] - met->lon[ix - 1],
00061
                                     met->lat[iy])));
             ls2 = gsl_pow_2(c * 500. * deg2dy(met->lat[iy + 1] - met->lat[iy - 1]));
00062
             if (fabs(met->lat[iy]) > 80)
00063
               ls2 *= (90. - fabs(met->lat[iy])) / 10.;
00064
00065
             k[ix][iy] = 1s2 * sqrt(2.0 * (gsl_pow_2(t) + gsl_pow_2(s)));
00066
00067
        /* Create output file... */
printf("Write data file: %s\n", argv[2]);
00068
00069
00070
        if (!(out = fopen(argv[2], "w")))
00071
          ERRMSG("Cannot create file!");
00072
00073
        /* Write header... */
00074
        fprintf(out,
    "# $1 = longitude [deg]\n"
00075
00076
                  "# $2 = latitude [deg]\n"
00077
                  "# $3 = zonal wind [m/s] \n"
00078
                  "# $4 = meridional wind [m/s]\n"
                  "# $5 = horizontal diffusivity [m^2/s]\n");
00079
00080
        /* Write data... */
for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
00081
00082
00083
           for (ix = 0; ix < met->nx; ix++)
00084
            00085
00086
00087
                        met->u[ix][iy][ip], met->v[ix][iy][ip], k[ix][iy]);
88000
00089
           for (ix = 0; ix < met->nx; ix++)
00090
            if (met->lon[ix] <= 180)</pre>
00091
               fprintf(out, "%g %g %g %g %g\n",
                        met->lon[ix], met->lat[iy],
00092
00093
                        \label{eq:met-variable} \text{met->u[ix][iy][ip], met->v[ix][iy][ip], k[ix][iy]);}
00094
00095
00096
         /* Close file... */
00097
        fclose(out);
00098
00099
        /* Free... */
00100
        free (met);
00101
00102
        return EXIT_SUCCESS;
00103 }
```

# 5.35 split.c File Reference

Split air parcels into a larger number of parcels.

#### **Functions**

• int main (int argc, char \*argv[])

### 5.35.1 Detailed Description

Split air parcels into a larger number of parcels.

Definition in file split.c.

#### 5.35.2 Function Documentation

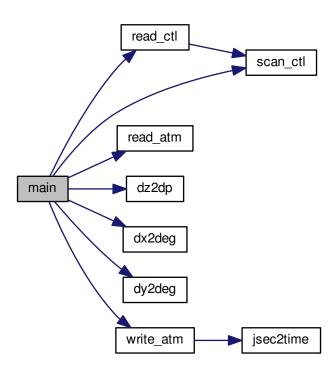
### 5.35.2.1 int main ( int argc, char \* argv[])

Definition at line 27 of file split.c.

```
00029
00030
00031
          atm_t *atm, *atm2;
00032
00033
          ctl t ctl;
00034
          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... */
00047
              (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
          to = scan_ctl(argv[1], argc, argv, "SPLIT_TO", -1, "0", NULL);
t1 = scan_ctl(argv[1], argc, argv, "SPLIT_T1", -1, "0", NULL);
00055
00056
          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);
00057
00058
00059
          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);
00061
00062
00063
00064
00065
00066
          /* Init random number generator... */
          gsl_rng_env_setup();
00067
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++) {
               mtot += atm->q[ctl.qnt_m][ip];
00076
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++) {
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);
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);
00092
00093
             /* Set time... */
00094
             if (t1 > t0)
00095
               atm2 - time[atm2 - np] = t0 + (t1 - t0) * qsl_rnq_uniform_pos(rnq);
00096
             else
00097
              atm2->time[atm2->np] = atm->time[ip]
00098
                 + gsl_ran_gaussian_ziggurat(rng, dt / 2.3548);
00099
00100
             /* Set vertical position... */
            if (z1 > z0)
00101
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]);
00106
            /* Set horizontal position... */
if (lon1 > lon0 && lat1 > lat0) {
00107
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
00111
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... */
            for (iq = 0; iq < ctl.nq; iq++)
  atm2->q[iq][atm2->np] = atm->q[iq][ip];
00119
00120
00121
           /* Adjust mass... */
if (ctl.qnt_m >= 0)
00122
00123
00124
              atm2->q[ctl.qnt_m][atm2->np] = mtot / n;
00125
           /* Increment particle counter... */
if ((++atm2->np) >= NP)
00126
00127
              ERRMSG("Too many air parcels!");
00128
00129
00130
00131
          /\star Save data and close file... \star/
00132
         write_atm(argv[3], &ctl, atm2, atm->time[0]);
00133
         /* Free... */
00134
00135
         free(atm);
00136
         free(atm2);
00137
00138
         return EXIT_SUCCESS;
00139 }
```

Here is the call graph for this function:



5.36 split.c 211

## 5.36 split.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) 2013-2018 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,
   t0, t1, z0, z1, lon0, lon1, lat0, lat1;
00037
00038
00039
00040
          int i, ip, iq, n;
00041
           /* Allocate... */
00042
          ALLOC(atm, atm_t, 1);
ALLOC(atm2, atm_t, 1);
00043
00044
00045
00046
          /* Check arguments... */
00047
          if (argc < 4)
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);
dr = scan_ctl(argv[1], argc, argv, "SPLIT_DT", -1, "0", NULL);
00051
00052
00053
00054
00055
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);
00058
00059
           dx = scan_ctl(argv[1], argc, argv, "SPLIT_DX", -1, "0", NULL);
00060
          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);
00061
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
           /* Get total and maximum mass... */
00074
          if (ctl.qnt_m >= 0)
            for (ip = 0; ip < atm->np; ip++) {
00075
               mtot += atm->q[ctl.qnt_m][ip];
00076
00077
                mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
00078
          if (m > 0)
00079
00080
            mtot = m:
00081
00082
           /* Loop over air parcels... */
          for (i = 0; i < n; i++) {
00083
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);
```

```
00090
00091
             ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00092
            /* Set time... */
00093
00094
           if (t1 > t0)
00095
             atm2 \rightarrow time[atm2 \rightarrow np] = t0 + (t1 - t0) * qsl_rnq_uniform_pos(rnq);
00096
            else
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 - p[atm2 - np] = P(z0 + (z1 - z0) * gsl_rng_uniform_pos(rng));
00103
00104
             atm2->p[atm2->np] = atm->p[ip]
00105
                + dz2dp(gsl\_ran\_gaussian\_ziggurat(rng, dz / 2.3548), atm->p[ip]);
00106
           /* Set horizontal position... */
if (lon1 > lon0 && lat1 > lat0) {
00107
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
00111
00112
             atm2 - lon[atm2 - lon[ip]] = atm - lon[ip]
             + 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
00118
           /* Copy quantities... */
           for (iq = 0; iq < ctl.nq; iq++)
  atm2->q[iq][atm2->np] = atm->q[iq][ip];
00119
00120
00121
00122
            /* Adjust mass...
00123
           if (ctl.qnt_m >= 0)
00124
             atm2->q[ctl.qnt_m][atm2->np] = mtot / n;
00125
          /* Increment particle counter... */
if ((++atm2->np) >= NP)
00126
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);
00136
        free(atm2);
00137
         return EXIT_SUCCESS;
00138
00139 }
```

## 5.37 time2jsec.c File Reference

Convert date to Julian seconds.

#### **Functions**

• int main (int argc, char \*argv[])

## 5.37.1 Detailed Description

Convert date to Julian seconds.

Definition in file time2jsec.c.

5.38 time2jsec.c 213

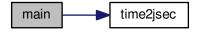
## 5.37.2 Function Documentation

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

Definition at line 27 of file time2jsec.c.

```
00029
00030
00031
        double jsec, remain;
00032
00033
        int day, hour, min, mon, sec, year;
00034
00035
         /* Check arguments... */
00036
          ERRMSG("Give parameters: <year> <mon> <day> <hour> <min> <sec> <remain>");
00037
00038
00039
        /* Read arguments... */
00040
        year = atoi(argv[1]);
00041
        mon = atoi(argv[2]);
00042
        day = atoi(argv[3]);
00043
        hour = atoi(argv[4]);
00044
        min = atoi(argv[5]);
sec = atoi(argv[6]);
00045
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;
00053 }
```

Here is the call graph for this function:



# 5.38 time2jsec.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.
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/>.
00016
00017
        Copyright (C) 2013-2018 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028
      int argc,
00029
        char *argv[]) {
```

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

#### 5.39 trac.c File Reference

Lagrangian particle dispersion model.

#### **Functions**

```
\bullet \ \ \text{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)

  Calculate mesoscale diffusion.
- void module\_diffusion\_turb (ctl\_t \*ctl, atm\_t \*atm, int ip, double dt, gsl\_rng \*rng)

Calculate turbulent 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[])

#### 5.39.1 Detailed Description

Lagrangian particle dispersion model.

Definition in file trac.c.

5.39 trac.c File Reference 215

# 5.39.2 Function Documentation

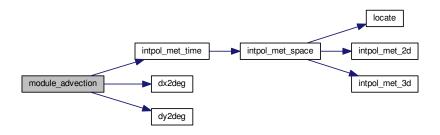
5.39.2.1 void module\_advection (  $met_t * met0$ ,  $met_t * met1$ ,  $atm_t * atm$ , int ip, double dt)

Calculate advection of air parcels.

Definition at line 383 of file trac.c.

```
00388
                    {
00389
00390
        double v[3], xm[3];
00391
        /* Interpolate meteorological data... */
00393
        intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
                        atm->lon[ip], atm->lat[ip], NULL, NULL, NULL, NULL,
00394
00395
                         &v[0], &v[1], &v[2], NULL, NULL, NULL);
00396
00397
        /* Get position of the mid point... */
        xm[0] = atm->lat[ip] + dx2deg(0.5 * dt * v[0] / 1000., atm->lat[ip]);
xm[1] = atm->lat[ip] + dy2deg(0.5 * dt * v[1] / 1000.);
00398
00399
00400
        xm[2] = atm->p[ip] + 0.5 * dt * v[2];
00401
        /\star Interpolate meteorological data for mid point... \star/
00402
        00403
00405
                         &v[0], &v[1], &v[2], NULL, NULL, NULL);
00406
00407
        /* Save new position... */
00408
        atm->time[ip] += dt;
atm->lon[ip] += dx2deg(dt * v[0] / 1000., xm[1]);
00409
00410
        atm->lat[ip] += dy2deg(dt * v[1] / 1000.);
00411
        atm \rightarrow p[ip] += dt * v[2];
00412 }
```

Here is the call graph for this function:



5.39.2.2 void module\_decay (  $ctl_t * ctl$ ,  $met_t * met0$ ,  $met_t * met1$ ,  $atm_t * atm$ , int ip, double dt)

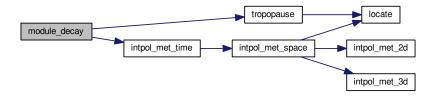
Calculate exponential decay of particle mass.

Definition at line 416 of file trac.c.

```
00422
                  {
00423
00424
       double ps, pt, tdec;
00425
00426
       /* Set constant lifetime... */
00427
       if (ctl->tdec_trop == ctl->tdec_strat)
00428
         tdec = ctl->tdec_trop;
00429
00430
       /* Set altitude-dependent lifetime... */
00431
       else {
```

```
00432
00433
          /* Get surface pressure... */
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00434
00435
                          atm->lon[ip], atm->lat[ip], &ps, NULL, NULL, NULL,
00436
                          NULL, NULL, NULL, NULL, NULL, NULL);
00437
00438
          /* Get tropopause pressure... */
00439
         pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00440
00441
          /* Set lifetime... */
00442
          if (atm->p[ip] <= pt)
00443
           tdec = ctl->tdec_strat;
00444
         else
00445
            tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
     p[ip]);
00446
00447
00448
        /* Calculate exponential decay... */
       atm->q[ctl->qnt_m][ip] *= exp(-dt / tdec);
00449
00450 }
```

Here is the call graph for this function:



5.39.2.3 void module\_diffusion\_meso ( ctl\_t \* ctl, met\_t \* met0, met\_t \* met1, atm\_t \* atm, int ip, double dt, gsl\_rng \* rng )

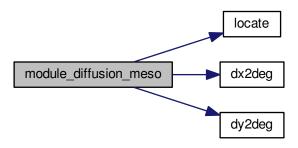
Calculate mesoscale diffusion.

Definition at line 454 of file trac.c.

```
00461
00462
00463
         double r, rs, u[16], v[16], w[16], usig, vsig, wsig;
00464
00465
         int ix, iy, iz;
00466
         /* Get indices... */
00467
         ix = locate(met0->lon, met0->nx, atm->lon[ip]);
00468
         iy = locate(met0->lat, met0->ny, atm->lat[ip]);
00469
00470
         iz = locate(met0->p, met0->np, atm->p[ip]);
00472
         /* Collect local wind data... */
00473
         u[0] = met0 \rightarrow u[ix][iy][iz];
         u[1] = met0->u[ix + 1][iy][iz];
00474
         u[2] = met0->u[ix][iy + 1][iz];
00475
         u[3] = met0->u[ix + 1][iy + 1][iz];
u[4] = met0->u[ix][iy][iz + 1];
00476
00477
00478
         u[5] = met0 -> u[ix + 1][iy][iz + 1];
00479
         u[6] = met0 -> u[ix][iy + 1][iz + 1];
         u[7] = met0 -> u[ix + 1][iy + 1][iz + 1];
00480
00481
         v[0] = met0 -> v[ix][iy][iz];
00482
00483
         v[1] = met0 -> v[ix + 1][iy][iz];
         v[2] = met0 -> v[ix][iy + 1][iz];
00484
         v[3] = met0->v[ix + 1][iy + 1][iz];
v[4] = met0->v[ix][iy][iz + 1];
00485
00486
        v[5] = met0->v[ix + 1][iy][iz + 1];
v[6] = met0->v[ix][iy + 1][iz + 1];
v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00487
00488
00489
00490
```

```
w[0] = met0->w[ix][iy][iz];
00492
         w[1] = met0->w[ix + 1][iy][iz];
00493
         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];
00494
00495
         w[5] = met0->w[ix + 1][iy][iz + 1];
w[6] = met0->w[ix][iy + 1][iz + 1];
00496
00498
         w[7] = met0 -> w[ix + 1][iy + 1][iz + 1];
00499
00500
         /* Get indices... */
         ix = locate(met1->lon, met1->nx, atm->lon[ip]);
00501
         iy = locate(met1->lat, met1->ny, atm->lat[ip]);
00502
00503
         iz = locate(met1->p, met1->np, atm->p[ip]);
00504
00505
         /* Collect local wind data... */
         u[8] = met1->u[ix][iy][iz];
u[9] = met1->u[ix + 1][iy][iz];
00506
00507
         u[10] = met1->u[ix][iy + 1][iz];
u[11] = met1->u[ix + 1][iy + 1][iz];
00508
00510
         u[12] = met1->u[ix][iy][iz + 1];
00511
         u[13] = met1->u[ix + 1][iy][iz + 1];
         u[14] = met1->u[ix][iy + 1][iz + 1];
00512
         u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00513
00514
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 + 1][iy + 1][iz];
00519
         v[12] = met1->v[ix][iy][iz + 1];
         v[13] = met1->v[ix + 1][iy][iz + 1];
v[14] = met1->v[ix][iy + 1][iz + 1];
00520
00521
00522
         v[15] = met1 - v[ix + 1][iy + 1][iz + 1];
00523
00524
         w[8] = met1->w[ix][iy][iz];
00525
         w[9] = met1->w[ix + 1][iy][iz];
         w[10] = met1->w[ix][iy + 1][iz];
00526
         w[11] = met1->w[ix + 1][iy + 1][iz];
w[12] = met1->w[ix][iy][iz + 1];
00527
00529
         w[13] = met1->w[ix + 1][iy][iz + 1];
00530
         w[14] = met1->w[ix][iy + 1][iz + 1];
00531
         w[15] = met1 -> w[ix + 1][iy + 1][iz + 1];
00532
         /\star Get standard deviations of local wind data... \star/
00533
         usig = gsl_stats_sd(u, 1, 16);
vsig = gsl_stats_sd(v, 1, 16);
00534
00535
00536
         wsig = gsl_stats_sd(w, 1, 16);
00537
00538
         /\star Set temporal correlations for mesoscale fluctuations... \star/
         r = 1 - 2 * fabs(dt) / ctl->dt_met;
00539
00540
         rs = sqrt(1 - r * r);
00541
00542
         /\star Calculate horizontal mesoscale wind fluctuations... \star/
00543
         if (ctl->turb_mesox > 0) {
00544
           atm->up[ip] = (float)
00545
              (r * atm->up[ip]
           + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_mesox * usig));
atm->lon[ip] += dx2deg(atm->up[ip] * dt / 1000., atm->lat[ip]);
00546
00547
00548
00549
           atm->vp[ip] = (float)
00550
              (r * atm->vp[ip]
           + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_mesox * vsig));
atm->lat[ip] += dy2deg(atm->vp[ip] * dt / 1000.);
00551
00552
00553
00554
00555
         /* Calculate vertical mesoscale wind fluctuations... */
00556
         if (ctl->turb_mesoz > 0) {
           atm->wp[ip] = (float)
00557
00558
              (r * atm->wp[ip]
00559
               + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_mesoz * wsig));
           atm->p[ip] += atm->wp[ip] * dt;
00560
00561
00562 }
```

Here is the call graph for this function:



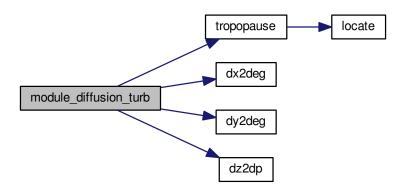
5.39.2.4 void module\_diffusion\_turb (  $ctl_t * ctl$ ,  $atm_t * atm$ , int ip, double dt,  $gsl_rng * rng$ )

Calculate turbulent diffusion.

Definition at line 566 of file trac.c.

```
00571
                         {
00572
00573
        double dx, dz, pt, p0, p1, w;
00574
00575
        /* Get tropopause pressure... */
00576
       pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00577
00578
        /* Get weighting factor... */
        p1 = pt * 0.866877899;
p0 = pt / 0.866877899;
00579
00580
        if (atm->p[ip] > p0)
00581
00582
          w = 1;
00583
        else if (atm->p[ip] < p1)</pre>
00584
          w = 0;
00585
        else
00586
          w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00587
        /* Set diffusivitiy... */
        dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00589
00590
00591
00592
        /* Horizontal turbulent diffusion... */
00593
        if (dx > 0) {
00594
          atm->lon[ip]
00595
            += dx2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00596
                        / 1000., atm->lat[ip]);
00597
          atm->lat[ip]
00598
             += dy2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00599
                        / 1000.);
00600
00601
00602
        /* Vertical turbulent diffusion... */
        if (dz > 0)
00603
00604
          atm->p[ip]
00605
             += dz2dp(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dz * fabs(dt)))
00606
                       / 1000., atm->p[ip]);
00607 }
```

Here is the call graph for this function:



5.39.2.5 void module\_isosurf (  $ctl_t * ctl$ ,  $met_t * met0$ ,  $met_t * met1$ ,  $atm_t * atm$ , int ip )

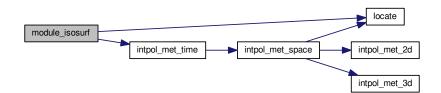
Force air parcels to stay on isosurface.

Definition at line 611 of file trac.c.

```
00617
00618
       static double *iso, *ps, t, *ts;
00619
00620
       static int idx, ip2, n;
00621
00622
       FILE *in;
00623
00624
       char line[LEN];
00625
00626
       /* Initialize... */
00627
       if (ip < 0) {</pre>
00628
00629
          /* Allocate... */
00630
         ALLOC(iso, double,
00631
               NP);
         ALLOC(ps, double, NP);
00632
00633
00634
         ALLOC(ts, double,
00635
               NP);
00636
00637
         /* Save pressure... */
if (ctl->isosurf == 1)
00638
           for (ip2 = 0; ip2 < atm->np; ip2++)
00639
             iso[ip2] = atm->p[ip2];
00640
00641
00642
         /\star Save density... \star/
         00643
00644
00645
00646
             st, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
iso[ip2] = atm->p[ip2] / t;
00647
00648
00649
00650
         /\star Save potential temperature... \star/
00651
         else if (ctl->isosurf == 3)
00652
00653
           for (ip2 = 0; ip2 < atm->np; ip2++) {
             00654
00655
00656
             &t, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
iso[ip2] = THETA(atm->p[ip2], t);
00657
00658
00659
```

```
00660
           /* Read balloon pressure data... */
00661
           else if (ctl->isosurf == 4) {
00662
00663
              /* Write info... */
00664
             printf("Read balloon pressure data: %s\n", ctl->balloon);
00665
00666
              /* Open file... */
00667
              if (!(in = fopen(ctl->balloon, "r")))
               ERRMSG("Cannot open file!");
00668
00669
00670
              /* Read pressure time series... */
             while (fgets(line, LEN, in))
  if (sscanf(line, "%lg %lg", &ts[n], &ps[n]) == 2)
    if ((++n) > NP)
00671
00672
00673
00674
                    ERRMSG("Too many data points!");
00675
00676
              /\star Check number of points... \star/
00677
             if (n < 1)
                ERRMSG("Could not read any data!");
00678
00679
00680
              /* Close file... */
00681
             fclose(in);
00682
           }
00683
00684
           /* Leave initialization... */
00685
           return;
00686
00687
00688
         /* Restore pressure... */
00689
         if (ctl->isosurf == 1)
00690
          atm->p[ip] = iso[ip];
00691
        /* Restore density... */
else if (ctl->isosurf == 2) {
00692
00693
00694
           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);
00695
00696
00697
           atm \rightarrow p[ip] = iso[ip] * t;
00698
00699
        /* Restore potential temperature... */
else if (ctl->isosurf == 3) {
00700
00701
00702
           intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00703
                             atm->lat[ip], NULL, NULL, NULL, &t,
          NULL, NULL, NULL, NULL, NULL, NULL); atm->p[ip] = 1000. * pow(iso[ip] / t, -1. / 0.286);
00704
00705
00706
00707
00708
         /* Interpolate pressure... */
00709
         else if (ctl->isosurf == 4) {
00710
         if (atm->time[ip] <= ts[0])</pre>
           atm->p[ip] = ps[0];
else if (atm->time[ip] >= ts[n - 1])
00711
00712
00713
             atm->p[ip] = ps[n - 1];
           00714
00715
             atm->p[ip] = LIN(ts[idx], ps[idx],
ts[idx + 1], ps[idx + 1], atm->time[ip]);
00716
00717
00718
00719
        }
00720 }
```

Here is the call graph for this function:



5.39.2.6 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 724 of file trac.c.

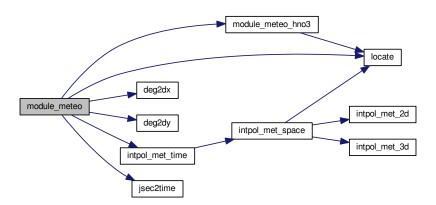
```
00729
                 {
00730
00731
        double a, b, c, ps, pt, pv, p_hno3, p_h2o, t, u, v, w, x1, x2, h2o, o3, z;
00732
00733
        /* Interpolate meteorological data... */
        intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00734
      lon[ip],
00735
                          atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &pv, &h2o, &o3);
00736
00737
        /* Set surface pressure... */
00738
        if (ctl->qnt_ps >= 0)
00739
          atm->q[ctl->qnt_ps][ip] = ps;
00740
00741
        /* Set tropopause pressure... */
00742
        if (ctl->qnt_pt >= 0)
00743
          atm->q[ctl->qnt_pt][ip] = pt;
00744
00745
        /* Set pressure... */
        if (ctl->qnt_p >= 0)
00746
00747
          atm \rightarrow q[ctl \rightarrow qnt_p][ip] = atm \rightarrow p[ip];
00748
        /\star Set geopotential height... \star/
00749
00750
        if (ctl->qnt_z >= 0)
00751
          atm \rightarrow q[ctl \rightarrow qnt_z][ip] = z;
00752
00753
        /* Set temperature... */
if (ctl->qnt_t >= 0)
00754
00755
          atm->q[ctl->qnt_t][ip] = t;
00756
00757
        /* Set zonal wind... */
00758
        if (ct.1->ant. u >= 0)
00759
          atm \rightarrow q[ctl \rightarrow qnt_u][ip] = u;
00760
00761
        /* Set meridional wind... */
00762
        if (ctl->qnt_v >= 0)
00763
          atm->q[ctl->qnt_v][ip] = v;
00764
00765
        /* Set vertical velocity... */
00766
        if (ctl->qnt_w >= 0)
00767
          atm->q[ctl->qnt_w][ip] = w;
00768
00769
        /* Set water vapor vmr... */
00770
        if (ct1->ant h2o >= 0)
00771
          atm->q[ctl->qnt_h2o][ip] = h2o;
00772
00773
        /* Set ozone vmr... */
00774
        if (ctl->qnt_o3 >= 0)
00775
          atm->q[ctl->qnt_o3][ip] = o3;
00776
00777
        /* Calculate horizontal wind... */
00778
        if (ctl->qnt_vh >= 0)
00779
          atm \rightarrow q[ctl \rightarrow qnt\_vh][ip] = sqrt(u * u + v * v);
00780
00781
         /* Calculate vertical velocity... */
        if (ctl->qnt_vz >= 0)
  atm->q[ctl->qnt_vz][ip] = -le3 * H0 / atm->p[ip] * w;
00782
00783
00784
00785
        /\star Calculate potential temperature... \star/
00786
        if (ctl->qnt_theta >= 0)
00787
          atm->q[ctl->qnt_theta][ip] = THETA(atm->p[ip], t);
00788
00789
        /* Set potential vorticity... */
00790
        if (ctl->qnt_pv >= 0)
00791
          atm->q[ctl->qnt_pv][ip] = pv;
00792
00793
         /* Calculate T_ice (Marti and Mauersberger, 1993)... */
00794
        if (ctl->qnt_tice >= 0)
00795
          atm->q[ctl->qnt_tice][ip] =
00796
            -2663.5 /
00797
             (log10((ct1->psc_h2o > 0 ? ct1->psc_h2o : h2o) * atm->p[ip] * 100.) -
00798
00799
00800
         /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
00801
        if (ctl->qnt_tnat >= 0) {
00802
          if (ctl->psc_hno3 > 0)
00803
            p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
00804
```

```
p_hno3 = clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip])
             * le-9 * atm->p[ip] / 1.333224;
p_h2o = (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
00806
00807
             a = 0.009179 - 0.00088 * log10(p_h2o);
b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
00808
00809
00810
             c = -11397.0 / a;
             x1 = (-b + \text{sqrt}(b * b - 4. * c)) / 2.;

x2 = (-b - \text{sqrt}(b * b - 4. * c)) / 2.;
00811
00812
00813
             if (x1 > 0)
             atm->q[ctl->qnt_tnat][ip] = x1;
if (x2 > 0)
00814
00815
00816
               atm->q[ctl->qnt_tnat][ip] = x2;
00817
00818
00819
           /* 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>
00820
00821
             ERRMSG("Need T_ice and T_NAT to calculate T_STS!");

atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
00822
00823
00824
                                                             + atm->q[ctl->qnt_tnat][ip]);
00825
00826 }
```

Here is the call graph for this function:



5.39.2.7 void module\_position ( met t \* met0, met t \* met1, atm\_t \* atm, int ip )

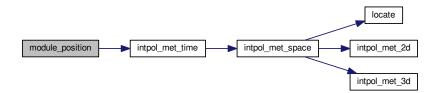
Check position of air parcels.

Definition at line 830 of file trac.c.

```
{
00835
00836
          double ps;
00837
00838
          /* Calculate modulo... */
          atm->lon[ip] = fmod(atm->lon[ip], 360);
atm->lat[ip] = fmod(atm->lat[ip], 360);
00839
00840
00841
          /* Check latitude... */
while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
00842
00843
           if (atm->lat[ip] > 90) {
   atm->lat[ip] = 180 - atm->lat[ip];
00844
00845
00846
               atm->lon[ip] += 180;
00847
00848
             if (atm->lat[ip] < -90) {</pre>
               atm->lat[ip] = -180 - atm->lat[ip];
atm->lon[ip] += 180;
00849
00850
00851
00852
00853
```

```
/* Check longitude... */
        while (atm->lon[ip] < -180)
  atm->lon[ip] += 360;
00855
00856
        while (atm->lon[ip] >= 180)
00857
00858
          atm->lon[ip] -= 360;
00859
00860
        /* Get surface pressure... */
00861
        intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00862
                          atm->lon[ip], atm->lat[ip], &ps, NULL, NULL, NULL,
00863
                          NULL, NULL, NULL, NULL, NULL, NULL);
00864
        /* Check pressure...
00865
        if (atm->p[ip] > ps)
00866
00867
          atm->p[ip] = ps;
00868
        else if (atm->p[ip] < met0->p[met0->np - 1])
00869
          atm \rightarrow p[ip] = met0 \rightarrow p[met0 \rightarrow np - 1];
00870 3
```

Here is the call graph for this function:



5.39.2.8 void module\_sedi ( ctl t \* ctl, met t \* met0, met t \* met1, atm t \* atm, int ip, double dt)

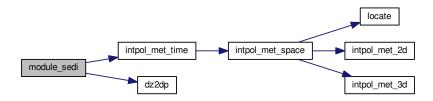
Calculate sedimentation of air parcels.

Definition at line 874 of file trac.c.

```
00880
00881
00882
        /* Coefficients for Cunningham slip-flow correction (Kasten, 1968): */
00883
        const double A = 1.249, B = 0.42, C = 0.87;
00884
00885
        /\star Average mass of an air molecule [kg/molec]: \star/
        const double m = 4.8096e-26;
00886
00887
00888
        double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
00889
00890
        /* Convert units... */
00891
        p = 100 * atm->p[ip];
        r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
00892
00893
        rho_p = atm->q[ctl->qnt_rho][ip];
00894
00895
        /* Get temperature... */
00896
        intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
                         atm->lat[ip], NULL, NULL, NULL, &T,
00897
00898
                         NULL, NULL, NULL, NULL, NULL, NULL);
00899
00900
        /* Density of dry air... */
00901
        rho = p / (RA * T);
00902
00903
        /* Dynamic viscosity of air... */
eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
00904
00905
00906
        /* Thermal velocity of an air molecule... */
00907
        v = sqrt(8 * KB * T / (M_PI * m));
00908
00909
        /* Mean free path of an air molecule... */
00910
        lambda = 2 * eta / (rho * v);
00911
00912
        /* Knudsen number for air... */
```

```
00913  K = lambda / r_p;
00914
00915  /* Cunningham slip-flow correction... */
00916  G = 1 + K * (A + B * exp(-C / K));
00917
00918  /* Sedimentation (fall) velocity... */
00919  v_p = 2. * gsl_pow_2(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;
00920
00921  /* Calculate pressure change... */
00922  atm->p[ip] += dz2dp(v_p * dt / 1000., atm->p[ip]);
00923 }
```

Here is the call graph for this function:



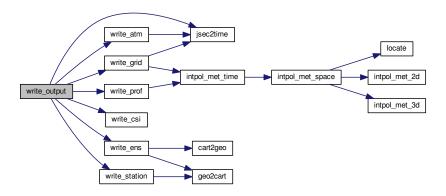
5.39.2.9 void write\_output ( const char \* dirname, ctl\_t \* ctl, met\_t \* met0, met\_t \* met1, atm\_t \* atm, double t)

Write simulation output.

Definition at line 927 of file trac.c.

```
00933
                     {
00935
         char filename[2 * LEN];
00936
00937
         double r;
00938
00939
         int year, mon, day, hour, min, sec;
00940
00941
00942
         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
00943
         /* Write atmospheric data... */
if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
    sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.tab",
00944
00945
00946
00947
                     dirname, ctl->atm_basename, year, mon, day, hour, min);
00948
           write_atm(filename, ctl, atm, t);
00949
00950
00951
         /* Write CSI data... */
         if (ctl->csi_basename[0] != '-') {
00952
           sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
00953
00954
            write_csi(filename, ctl, atm, t);
00955
00956
         /* Write ensemble data... */
if (ctl->ens_basename[0] != '-') {
00957
00958
          sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
write_ens(filename, ctl, atm, t);
00959
00960
00961
00962
         /* Write gridded data... */
if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
00963
00964
           sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
00965
00966
                    dirname, ctl->grid_basename, year, mon, day, hour, min);
00967
            write_grid(filename, ctl, met0, met1, atm, t);
00968
00969
         /* Write profile data... */
if (ctl->prof_basename[0] != '-') {
00970
00971
00972
           sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
```

Here is the call graph for this function:



# 5.39.2.10 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
        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... */
00143
        if (argc < 5)</pre>
          ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00144
00145
        /* Open directory list... */
if (!(dirlist = fopen(argv[1], "r")))
00146
00147
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)
```

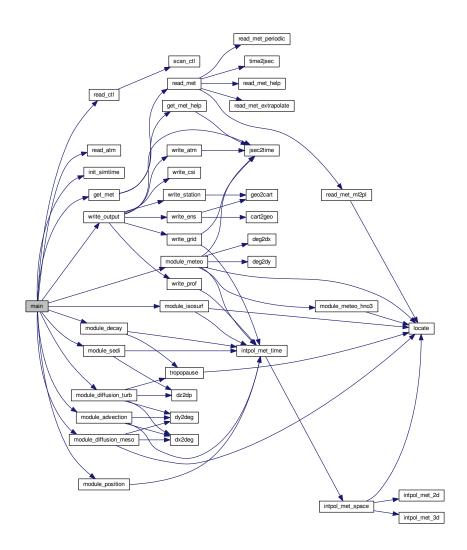
```
continue;
00156
00157
00158
              Initialize model run...
00159
00160
00161
           /* Set timers... */
00162
           START_TIMER(TIMER_TOTAL);
00163
           START_TIMER (TIMER_INIT);
00164
00165
           /* Allocate... */
          ALLOC(atm, atm_t, 1);
ALLOC(met0, met_t, 1);
00166
00167
00168
           ALLOC(met1, met_t, 1);
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
           /* Read control parameters... */
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]);
read_atm(filename, &ctl, atm);
00186
00187
00188
           /* Set start time...
00189
           if (ctl.direction == 1) {
00190
00191
            ctl.t_start = gsl_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);
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
           /\star Check time interval... \star/
00200
00201
           if (ctl.direction * (ctl.t_stop - ctl.t_start) <= 0)</pre>
00202
            ERRMSG("Nothing to do!");
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
             /* Set time steps for air parcels... */
             for (ip = 0; ip < atm->np; ip++)
  if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
00226
00227
                    && ctl.direction * (atm->time[ip] - ctl.t_stop) <= 0
&& ctl.direction * (atm->time[ip] - t) < 0))
00228
00229
00230
                 dt[ip] = t - atm->time[ip];
00231
               else
                 dt[ip] = GSL_NAN;
00232
00233
00234
             /* Get meteorological data... */
00235
             START_TIMER(TIMER_INPUT);
00236
             get_met(&ctl, argv[4], t, met0, met1);
             if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00237
               printf("Warning: Violation of CFL criterion! Set DT_MOD <= %g s!\n",
00238
```

```
fabs (met 0 - > lon[1] - met 0 - > lon[0]) * 111132. / 150.);
00240
             STOP_TIMER(TIMER_INPUT);
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);
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++)
00251
00252
              if (gsl_finite(dt[ip]))
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],
                                           rng[omp_get_thread_num()]);
00264
00265
00266
             STOP_TIMER(TIMER_DIFFTURB);
00267
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)
00272
              for (ip = 0; ip < atm->np; ip++)
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
             /* 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)
              for (ip = 0; ip < atm->np; ip++)
00284
                 if (gsl_finite(dt[ip]))
00285
                   module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
00286
             STOP TIMER (TIMER SEDI):
00287
00288
             /* Isosurface... */
00290
             START_TIMER(TIMER_ISOSURF);
00291 if (ctl.isosurf >= 1 && ctl.isosurf <= 4) {
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... */
             START_TIMER(TIMER_POSITION);
00299
00300 #pragma omp parallel for default(shared) private(ip)
00301
            for (ip = 0; ip < atm->np; ip++)
00302
               module_position(met0, met1, atm, ip);
00303
             STOP_TIMER(TIMER_POSITION);
00304
00305
             /* Meteorological data... */
             START_TIMER(TIMER_METEO);
00306
00307 #pragma omp parallel for default(shared) private(ip)
            for (ip = 0; ip < atm->np; ip++)
00309
               module_meteo(&ctl, met0, met1, atm, ip);
00310
             STOP_TIMER(TIMER_METEO);
00311
00312
             /* Decav... */
             START_TIMER (TIMER_DECAY);
00313
             if ((ctl.tdec_trop > 0 || ctl.tdec_strat > 0) && ctl.
      qnt_m >= 0) {
00315 #pragma omp parallel for default(shared) private(ip)
               for (ip = 0; ip < atm->np; ip++)
if (gsl_finite(dt[ip]))
00316
00317
00318
                   module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00319
00320
             STOP_TIMER(TIMER_DECAY);
00321
00322
             /* Write output... */
             START_TIMER (TIMER OUTPUT):
00323
00324
             write_output(dirname, &ctl, met0, met1, atm, t);
```

```
00325
            STOP_TIMER(TIMER_OUTPUT);
00326
00327
00328
00329
            Finalize model run...
00330
00331
00332
          /* Report memory usage... */
          00333
00334
00335
00336
          00337
00338
00339
00340
00341
00342
          /* Report problem size... */
printf("SIZE_NP = %d\n", atm->np);
printf("SIZE_TASKS = %d\n", size);
00343
00344
00345
          printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00346
          /* Report timers... */
00347
          STOP_TIMER(TIMER_TOTAL);
PRINT_TIMER(TIMER_TOTAL);
PRINT_TIMER(TIMER_INIT);
00348
00349
00350
00351
          PRINT_TIMER(TIMER_INPUT);
00352
          PRINT_TIMER(TIMER_OUTPUT);
00353
          PRINT_TIMER (TIMER_ADVECT);
          PRINT_TIMER(TIMER_DECAY);
PRINT_TIMER(TIMER_DIFFMESO);
00354
00355
00356
          PRINT_TIMER (TIMER_DIFFTURB);
00357
          PRINT_TIMER(TIMER_ISOSURF);
00358
          PRINT_TIMER(TIMER_METEO);
00359
          PRINT_TIMER(TIMER_POSITION);
00360
          PRINT_TIMER (TIMER_SEDI);
00361
00362
          /* Free random number generators... */
00363
          for (i = 0; i < NTHREADS; i++)</pre>
00364
           gsl_rng_free(rng[i]);
00365
00366
          /* Free... */
00367
          free (atm):
00368
          free (met0);
00369
          free (met1);
00370
          free(dt);
00371
00372
00373 #ifdef MPI
00374  /* Finalize MPI... */
00375  MPI_Finalize();
00376 #endif
00377
00378
       return EXIT_SUCCESS;
00379 }
```

5.40 trac.c 229

Here is the call graph for this function:



# 5.40 trac.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.
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) 2013-2018 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,
         atm_t * atm,
00048
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(
        ctl_t * ctl,
met_t * met0,
00072
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
00107
        met_t * met1,
        atm_t * atm,
00108
00109
         double t);
00110
00111 /* -----
         Main...
00112
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
00125
         gsl_rng *rng[NTHREADS];
00126
```

5.40 trac.c 231

```
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... */
        MPI_Init(&argc, &argv);
00137
        MPI_Comm_rank (MPI_COMM_WORLD, &rank);
00138
00139
        MPI_Comm_size(MPI_COMM_WORLD, &size);
00140 #endif
00141
00142
         /* Check arguments... */
00143
        if (argc < 5)
          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
00169
           ALLOC(dt, double,
                 NP);
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
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.
      dt_mod;
00209
           /* Set timers... */
00210
00211
           STOP_TIMER(TIMER_INIT);
```

```
00212
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
     dt_mod;
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.)
             printf("Warning: Violation of CFL criterion! Set DT_MOD <= %g s!\n",</pre>
00238
                     fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.);
00239
            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
            STOP_TIMER(TIMER_ISOSURF);
00246
00247
            /* Advection...
00249
            START_TIMER(TIMER_ADVECT);
00250 #pragma omp parallel for default(shared) private(ip)
00251
            for (ip = 0; ip < atm->np; ip++)
00252
             if (gsl_finite(dt[ip]))
                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++)
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);
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
                  module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
00285
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
            STOP TIMER (TIMER ISOSURF);
00296
00297
```

5.40 trac.c 233

```
/* Position...
             START_TIMER(TIMER_POSITION);
00299
00300 #pragma omp parallel for default(shared) private(ip)
00301
           for (ip = 0; ip < atm->np; ip++)
               module_position(met0, met1, atm, ip);
00302
            STOP_TIMER(TIMER_POSITION);
00303
00305
             /* Meteorological data... */
00306
            START_TIMER(TIMER_METEO);
00307 \#pragma omp parallel for default(shared) private(ip)
            for (ip = 0; ip < atm->np; ip++)
  module_meteo(&ctl, met0, met1, atm, ip);
00308
00309
00310
            STOP_TIMER(TIMER_METEO);
00311
00312
             /* Decay... */
00313
             START_TIMER(TIMER_DECAY);
00314
             if ((ctl.tdec_trop > 0 || ctl.tdec_strat > 0) && ctl.
      qnt m >= 0) {
00315 #pragma omp parallel for default(shared) private(ip)
00316
              for (ip = 0; ip < atm->np; ip++)
00317
                if (gsl_finite(dt[ip]))
00318
                   module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00319
             STOP_TIMER(TIMER DECAY):
00320
00321
00322
             /* Write output...
00323
             START_TIMER(TIMER_OUTPUT);
00324
            write_output(dirname, &ctl, met0, met1, atm, t);
00325
             STOP_TIMER(TIMER_OUTPUT);
00326
00327
00328
00329
             Finalize model run...
00330
00331
00332
           /* 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",
00333
00334
00335
00336
                  4 * NP * sizeof(double) / 1024. / 1024.);
          00337
00338
00339
                   + (GX * GY + GX * GY * GZ) * sizeof(int)) / 1024. / 1024.);
00340
00341
00342
           /* Report problem size... */
          printf("SIZE_NP = %d\n", atm->np);
printf("SIZE_TASKS = %d\n", size);
printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00343
00344
00345
00346
00347
           /* Report timers...
00348
           STOP_TIMER(TIMER_TOTAL);
00349
           PRINT_TIMER (TIMER_TOTAL);
00350
           PRINT_TIMER(TIMER_INIT);
          PRINT_TIMER (TIMER_INPUT);
00351
          PRINT_TIMER(TIMER_OUTPUT);
PRINT_TIMER(TIMER_ADVECT);
00352
00354
           PRINT_TIMER (TIMER_DECAY);
00355
           PRINT_TIMER (TIMER_DIFFMESO);
00356
           PRINT_TIMER(TIMER_DIFFTURB);
          PRINT_TIMER(TIMER_ISOSURF);
00357
          PRINT_TIMER (TIMER_METEO);
00358
00359
           PRINT_TIMER (TIMER_POSITION);
00360
          PRINT_TIMER(TIMER_SEDI);
00361
          /* Free random number generators... */
for (i = 0; i < NTHREADS; i++)</pre>
00362
00363
            gsl_rng_free(rng[i]);
00364
00365
00366
           /* Free... */
00367
           free(atm);
00368
           free (met0);
           free (met1);
00369
00370
          free(dt);
00371
00372
00373 #ifdef MPI
00374 /* Finalize MPI... */
00375 MPT Finalize (
00376 #endif
00377
00378
        return EXIT_SUCCESS;
00379 }
00380
00382
00383 void module advection(
```

```
00384
       met_t * met0,
       met_t * met1,
atm_t * atm,
00385
00386
00387
       int ip,
00388
       double dt) {
00389
00390
       double v[3], xm[3];
00391
00392
        /* Interpolate meteorological data... */
       00393
00394
00395
                       &v[0], &v[1], &v[2], NULL, NULL, NULL);
00396
00397
        /* Get position of the mid point... */
       00398
00399
00400
       xm[2] = atm->p[ip] + 0.5 * dt * v[2];
00401
00402
        /* Interpolate meteorological data for mid point... */
00403
       intpol_met_time(met0, met1, atm->time[ip] + 0.5 * dt,
                       xm[2], xm[0], xm[1], NULL, NULL, NULL, NULL, &v[0], &v[1], &v[2], NULL, NULL, NULL);
00404
00405
00406
00407
       /* Save new position... */
       atm->time[ip] += dt;
atm->lon[ip] += dx2deg(dt * v[0] / 1000., xm[1]);
atm->lat[ip] += dy2deg(dt * v[1] / 1000.);
00408
00410
00411
       atm->p[ip] += dt * v[2];
00412 }
00413
00415
00416 void module_decay(
00417
       ctl_t * ctl,
00418
       met_t * met0,
       met_t * met1,
atm_t * atm,
00419
00420
00421
       int ip,
00422
       double dt) {
00423
00424
       double ps, pt, tdec;
00425
00426
       /* Set constant lifetime... */
00427
       if (ctl->tdec_trop == ctl->tdec_strat)
         tdec = ctl->tdec_trop;
00428
00429
00430
       /* Set altitude-dependent lifetime... */
       else {
00431
00432
         /* Get surface pressure... */
intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00433
00434
00435
                         atm->lon[ip], atm->lat[ip], &ps, NULL, NULL, NULL,
00436
                         NULL, NULL, NULL, NULL, NULL, NULL);
00437
         /\star Get tropopause pressure... \star/
00438
00439
         pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00440
00441
         /* Set lifetime... */
00442
         if (atm->p[ip] \ll pt)
00443
           tdec = ctl->tdec_strat;
00444
         else
           tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
00445
     p[ip]);
00446
00447
00448
        /* Calculate exponential decay... */
       atm \rightarrow q[ctl \rightarrow qnt_m][ip] *= exp(-dt / tdec);
00449
00450 }
00451
00453
00454 void module_diffusion_meso(
       ctl_t * ctl,
met_t * met0,
00455
00456
       met_t * met1,
00457
00458
       atm_t * atm,
00459
        int ip,
00460
       double dt,
00461
       gsl_rng * rng) {
00462
00463
       double r, rs, u[16], v[16], w[16], usig, vsig, wsig;
00464
00465
       int ix, iy, iz;
00466
       /* Get indices... */
ix = locate(met0->lon, met0->nx, atm->lon[ip]);
00467
00468
       iy = locate(met0->lat, met0->ny, atm->lat[ip]);
00469
```

5.40 trac.c 235

```
iz = locate(met0->p, met0->np, atm->p[ip]);
00471
00472
         /* Collect local wind data... */
         u[0] = met0->u[ix][iy][iz];
00473
         u[1] = met0->u[ix + 1][iy][iz];
00474
        u[2] = met0->u[ix][iy + 1][iz];
u[3] = met0->u[ix + 1][iy + 1][iz];
00475
00477
         u[4] = met0 -> u[ix][iy][iz + 1];
00478
         u[5] = met0 -> u[ix + 1][iy][iz + 1];
00479
         u[6] = met0->u[ix][iy + 1][iz + 1];
         u[7] = met0 -> u[ix + 1][iy + 1][iz + 1];
00480
00481
00482
         v[0] = met0 -> v[ix][iy][iz];
         v[1] = met0 -> v[ix + 1][iy][iz];
00483
00484
         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];
00485
00486
        v[5] = met0 > v[ix][iy][i2 + 1];
v[5] = met0 -> v[ix + 1][iy][iz + 1];
v[6] = met0 -> v[ix][iy + 1][iz + 1];
00487
00488
00489
         v[7] = met0 -> v[ix + 1][iy + 1][iz + 1];
00490
00491
         w[0] = met0 -> w[ix][iy][iz];
        w[0] = met0->w[ix + 1][iy][iz];
w[2] = met0->w[ix][iy + 1][iz];
w[3] = met0->w[ix][iy + 1][iz];
w[4] = met0->w[ix][iy][iz + 1];
00492
00493
00494
00496
         w[5] = met0 -> w[ix + 1][iy][iz + 1];
00497
         w[6] = met0->w[ix][iy + 1][iz + 1];
00498
         w[7] = met0 -> w[ix + 1][iy + 1][iz + 1];
00499
00500
         /* Get indices... */
00501
         ix = locate(met1->lon, met1->nx, atm->lon[ip]);
00502
         iy = locate(met1->lat, met1->ny, atm->lat[ip]);
00503
         iz = locate(met1->p, met1->np, atm->p[ip]);
00504
00505
         /* Collect local wind data... */
        u[8] = met1->u[ix][iy][iz];
u[9] = met1->u[ix + 1][iy][iz];
00506
00508
         u[10] = met1->u[ix][iy + 1][iz];
00509
         u[11] = met1->u[ix + 1][iy + 1][iz];
00510
         u[12] = met1->u[ix][iy][iz + 1];
        u[13] = met1->u[ix + 1][iy][iz + 1];
u[14] = met1->u[ix][iy + 1][iz + 1];
00511
00512
00513
         u[15] = met1 -> u[ix + 1][iy + 1][iz + 1];
00514
         v[8] = met1->v[ix][iy][iz];
00515
00516
        v[9] = met1->v[ix + 1][iy][iz];
00517
         v[10] = met1->v[ix][iy + 1][iz];
        v[11] = met1->v[ix,[iy, i],[iz],
v[11] = met1->v[ix + 1][iy + 1][iz];
v[12] = met1->v[ix][iy][iz + 1];
00518
00519
         v[13] = met1 - v[ix + 1][iy][iz + 1];
00521
         v[14] = met1->v[ix][iy + 1][iz + 1];
00522
         v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00523
         w[8] = met1->w[ix][iy][iz];
00524
         w[9] = met1->w[ix + 1][iy][iz];
w[10] = met1->w[ix][iy + 1][iz];
00525
         w[11] = met1->w[ix + 1][iy + 1][iz];
00527
00528
         w[12] = met1->w[ix][iy][iz + 1];
00529
         w[13] = met1->w[ix + 1][iy][iz + 1];
         w[14] = met1->w[ix][iy + 1][iz + 1];
00530
         w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00531
00532
00533
         /* Get standard deviations of local wind data... */
         usig = gsl_stats_sd(u, 1, 16);
00534
         vsig = gsl_stats_sd(v, 1, 16);
00535
00536
         wsig = gsl_stats_sd(w, 1, 16);
00537
00538
         /* Set temporal correlations for mesoscale fluctuations... */
         r = 1 - 2 * fabs(dt) / ctl->dt_met;
00540
         rs = sqrt(1 - r * r);
00541
00542
         /\star Calculate horizontal mesoscale wind fluctuations... \star/
00543
         if (ctl->turb_mesox > 0) {
           atm->up[ip] = (float)
00544
00545
             (r * atm->up[ip]
00546
               + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_mesox * usig));
00547
           atm \rightarrow lon[ip] += dx2deg(atm \rightarrow up[ip] * dt / 1000., atm \rightarrow lat[ip]);
00548
00549
           atm->vp[ip] = (float)
             (r * atm->vp[ip]
00550
               + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_mesox * vsig));
00552
           atm->lat[ip] += dy2deg(atm->vp[ip] * dt / 1000.);
00553
00554
00555
         /* Calculate vertical mesoscale wind fluctuations... */
00556
         if (ctl->turb_mesoz > 0) {
```

```
atm->wp[ip] = (float)
00558
           (r * atm->wp[ip]
00559
            + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_mesoz * wsig));
00560
         atm->p[ip] += atm->wp[ip] * dt;
00561
00562 }
00563
00565
00566 void module_diffusion_turb(
00567
       ctl_t * ctl,
atm_t * atm,
00568
00569
       int ip,
00570
       double dt,
00571
       gsl_rng * rng) {
00572
00573
       double dx, dz, pt, p0, p1, w;
00574
00575
       /* Get tropopause pressure... */
00576
       pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00577
00578
       /* Get weighting factor... */
       p1 = pt * 0.866877899;
p0 = pt / 0.866877899;
00579
00580
00581
       if (atm->p[ip] > p0)
00582
         w = 1;
00583
       else if (atm->p[ip] < p1)
00584
         w = 0;
00585
       else
         w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00586
00587
00588
        /* Set diffusivitiy... */
       dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00589
00590
00591
00592
       /* Horizontal turbulent diffusion... */
00593
       if (dx > 0) {
00594
         atm->lon[ip]
00595
           += dx2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00596
                      / 1000., atm->lat[ip]);
00597
         atm->lat[ip]
00598
           += dy2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00599
                     / 1000.):
00600
00601
00602
        /* Vertical turbulent diffusion... */
00603
       if (dz > 0)
00604
         atm->p[ip]
00605
           += dz2dp(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dz * fabs(dt)))
00606
                    / 1000., atm->p[ip]);
00607 }
00608
00610
00611 void module isosurf(
00612
       ctl_t * ctl,
met_t * met0,
00613
00614
       met_t * met1,
       atm_t * atm,
00615
00616
       int ip) {
00617
       static double *iso, *ps, t, *ts;
00618
00619
00620
       static int idx, ip2, n;
00621
00622
       FILE *in;
00623
00624
       char line[LEN];
00625
00626
       /* Initialize... */
00627
       if (ip < 0) {</pre>
00628
00629
          /* Allocate... */
00630
         ALLOC(iso, double,
               NP);
00631
00632
         ALLOC(ps, double,
00633
               NP);
00634
         ALLOC(ts, double,
00635
               NP);
00636
00637
         /* Save pressure... */
         if (ctl->isosurf == 1)
00638
00639
           for (ip2 = 0; ip2 < atm->np; ip2++)
00640
             iso[ip2] = atm->p[ip2];
00641
00642
         /* Save density... */
         else if (ctl->isosurf == 2)
00643
```

5.40 trac.c 237

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

ERRMSG("Too many data points!");
00673
00674
00675
00676
           /\star Check number of points... \star/
00677
00678
             ERRMSG("Could not read any data!");
00679
           /* Close file... */
00680
00681
           fclose(in);
00682
00683
00684
         /* Leave initialization... */
00685
         return;
00686
00687
       /* Restore pressure... */
00689
       if (ctl->isosurf == 1)
00690
         atm->p[ip] = iso[ip];
00691
00692
       /* Restore density... */
else if (ctl->isosurf == 2) {
00693
        intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00694
     lon[ip],
00695
                         atm->lat[ip], NULL, NULL, NULL, &t,
00696
                         NULL, NULL, NULL, NULL, NULL, NULL);
         atm \rightarrow p[ip] = iso[ip] * t;
00697
00698
00700
        /* Restore potential temperature... */
00701
       else if (ctl->isosurf == 3) {
00702
         intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
     lon[ip],
         00703
00704
00705
00706
00707
       /* Interpolate pressure... */
00708
       else if (ctl->isosurf == 4) {
  if (atm->time[ip] <= ts[0])</pre>
00709
00710
           atm->p[ip] = ps[0];
00711
         else if (atm->time[ip] >= ts[n - 1])
00712
00713
           atm->p[ip] = ps[n - 1];
00714
         else {
           00715
00716
00717
00718
00719
00720 }
00721
00723
00724 void module_meteo(
00725
      ctl_t * ctl,
00726
       met_t * met0,
       met_t * met1,
atm_t * atm,
00727
00728
```

```
00729
        int ip) {
00730
00731
        double a, b, c, ps, pt, pv, p_hno3, p_h2o, t, u, v, w, x1, x2, h2o, o3, z;
00732
00733
        /* Interpolate meteorological data... */
        intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00734
      lon[ip],
00735
                          atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &pv, &h2o, &o3);
00736
00737
         /* Set surface pressure... */
        if (ctl->qnt_ps >= 0)
  atm->q[ctl->qnt_ps][ip] = ps;
00738
00739
00740
00741
         /* Set tropopause pressure... */
00742
        if (ctl->qnt_pt >= 0)
00743
          atm->q[ctl->qnt_pt][ip] = pt;
00744
00745
        /* Set pressure... */
00746
        if (ctl->qnt_p >= 0)
00747
          atm->q[ctl->qnt_p][ip] = atm->p[ip];
00748
00749
        /* Set geopotential height... */
00750
        if (ctl->qnt_z >= 0)
00751
          atm \rightarrow q[ctl \rightarrow qnt_z][ip] = z;
00752
00753
        /* Set temperature... */
00754
        if (ctl->qnt_t >= 0)
00755
          atm \rightarrow q[ctl \rightarrow qnt_t][ip] = t;
00756
00757
        /* Set zonal wind... */
00758
        if (ctl->gnt u >= 0)
00759
          atm->q[ctl->qnt_u][ip] = u;
00760
00761
        /\star Set meridional wind... \star/
00762
        if (ctl->qnt_v >= 0)
00763
          atm->q[ctl->qnt_v][ip] = v;
00764
00765
        /* Set vertical velocity... */
00766
        if (ctl->qnt_w >= 0)
00767
          atm \rightarrow q[ctl \rightarrow qnt_w][ip] = w;
00768
00769
        /* Set water vapor vmr... */
00770
        if (ctl->qnt_h2o >= 0)
00771
          atm->q[ctl->qnt_h2o][ip] = h2o;
00772
00773
        /* Set ozone vmr...
00774
        if (ct1->qnt_o3 >= 0)
00775
          atm->q[ctl->qnt_o3][ip] = o3;
00776
00777
        /* Calculate horizontal wind... */
00778
        if (ctl->qnt_vh >= 0)
00779
          atm->q[ctl->qnt\_vh][ip] = sqrt(u * u + v * v);
00780
00781
         /* Calculate vertical velocity... */
00782
        if (ctl->ant vz >= 0)
00783
          atm \rightarrow q[ctl \rightarrow qnt_vz][ip] = -1e3 * H0 / atm \rightarrow p[ip] * w;
00784
00785
        /* Calculate potential temperature... */
00786
        if (ctl->qnt_theta >= 0)
00787
          atm->q[ctl->qnt_theta][ip] = THETA(atm->p[ip], t);
00788
00789
        /* Set potential vorticity... */
00790
        if (ctl->qnt_pv >= 0)
00791
          atm->q[ctl->qnt_pv][ip] = pv;
00792
00793
         /* Calculate T_ice (Marti and Mauersberger, 1993)... */
        if (ctl->qnt_tice >= 0)
00794
          atm->q[ctl->qnt_tice][ip] =
00795
00796
             -2663.5 /
00797
             (\log 10((ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] * 100.) -
00798
00799
00800
         /\star Calculate T_NAT (Hanson and Mauersberger, 1988)... \star/
        if (ctl->qnt_tnat >= 0) {
   if (ctl->psc_hno3 > 0)
00801
00802
            p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
00803
00804
00805
           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;
00806
00807
          b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
00808
00809
00810
           c = -11397.0 / a;
00811
           x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
           x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
00812
           if (x1 > 0)
00813
00814
             atm->g[ctl->gnt tnat][ip] = x1;
```

5.40 trac.c 239

```
if (x2 > 0)
00816
           atm->q[ctl->qnt_tnat][ip] = x2;
00817
00818
00819
        /* 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>
00820
00822
            ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
00823
          atm -> q[ctl -> qnt\_tsts][ip] = 0.5 * (atm -> q[ctl -> qnt\_tice][ip]
00824
                                               + atm->q[ctl->qnt_tnat][ip]);
00825
00826 }
00827
00829
00830 void module_position(
       met_t * met0,
met_t * met1,
atm_t * atm,
00831
00832
00833
00834
        int ip) {
00835
00836
        double ps;
00837
       /* Calculate modulo... */
atm->lon[ip] = fmod(atm->lon[ip], 360);
00838
00839
        atm->lat[ip] = fmod(atm->lat[ip], 360);
00841
        /* Check latitude... */
while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
00842
00843
         if (atm->lat[ip] > 90) {
  atm->lat[ip] = 180 - atm->lat[ip];
  atm->lon[ip] += 180;
00844
00845
00846
00847
00848
          if (atm->lat[ip] < -90) {
            atm->lat[ip] = -180 - atm->lat[ip];
atm->lon[ip] += 180;
00849
00850
00851
00852
00853
00854
        /* Check longitude... */
00855
        while (atm->lon[ip] < -180)
00856
         atm->lon[ip] += 360;
00857
        while (atm->lon[ip] >= 180)
00858
         atm->lon[ip] -= 360;
00859
00860
        /* Get surface pressure... */
00861
        intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
                         atm->lon[ip], atm->lat[ip], &ps, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
00862
00863
00864
00865
        /* Check pressure... */
00866
       if (atm->p[ip] > ps)
00867
          atm->p[ip] = ps;
00868
        else if (atm->p[ip] < met0->p[met0->np - 1])
00869
          atm -> p[ip] = met0 -> p[met0 -> np - 1];
00870 }
00871
00873
00874 void module_sedi(
00875
        ctl_t * ctl,
met_t * met0,
00876
00877
        met_t * met1,
00878
        atm_t * atm,
        int ip,
00879
00880
        double dt) {
00881
00882
        /* Coefficients for Cunningham slip-flow correction (Kasten, 1968): */
        const double A = 1.249, B = 0.42, C = 0.87;
00883
00884
00885
        /* Average mass of an air molecule [kg/molec]: */
00886
        const double m = 4.8096e-26;
00887
00888
        double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
00889
00890
        /* Convert units... */
00891
        p = 100 * atm->p[ip];
00892
        r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
00893
        rho_p = atm->q[ctl->qnt_rho][ip];
00894
        /* Get temperature... */
00895
00896
        intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00897
                         atm->lat[ip], NULL, NULL, NULL, &T,
00898
                         NULL, NULL, NULL, NULL, NULL, NULL);
00899
00900
        /* Density of dry air... */
```

```
00901
       rho = p / (RA * T);
00902
        /* Dynamic viscosity of air... */ eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
00903
00904
00905
00906
        /* Thermal velocity of an air molecule... */
        v = sqrt(8 * KB * T / (M_PI * m));
00908
00909
        /* Mean free path of an air molecule... */
00910
        lambda = 2 * eta / (rho * v);
00911
00912
        /* Knudsen number for air... */
00913
        K = lambda / r p;
00914
00915
        /\star Cunningham slip-flow correction... \star/
00916
        G = 1 + K * (A + B * exp(-C / K));
00917
        /* Sedimentation (fall) velocity... */ v_p = 2. * gsl_pow_2(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;
00918
00919
00920
        /* Calculate pressure change... */ atm->p[ip] += dz2dp(v_p * dt / 1000., atm->p[ip]);
00921
00922
00923 }
00924
00926
00927 void write_output(
00928
       const char *dirname,
        ctl_t * ctl,
met_t * met0,
00929
00930
00931
        met t * met1.
00932
        atm_t * atm,
00933
        double t) {
00934
00935
        char filename[2 * LEN];
00936
00937
        double r;
00938
00939
        int year, mon, day, hour, min, sec;
00940
00941
        / * \ \mathsf{Get} \ \mathsf{time} \ldots \ * /
00942
        jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
00943
        /* Write atmospheric data... */
if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
00944
00945
         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
00946
00947
                  dirname, ctl->atm_basename, year, mon, day, hour, min);
00948
          write_atm(filename, ctl, atm, t);
00949
00950
00951
        /* Write CSI data... */
00952
        if (ctl->csi_basename[0] != '-') {
00953
         sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
00954
          write_csi(filename, ctl, atm, t);
00955
00956
        /* Write ensemble data... */
        if (ctl->ens_basename[0] != '-') {
00958
00959
         sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
00960
          write_ens(filename, ctl, atm, t);
00961
00962
00963
        /* Write gridded data...
        if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
00964
00965
         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
00966
                  dirname, ctl->grid_basename, year, mon, day, hour, min);
00967
          write_grid(filename, ctl, met0, met1, atm, t);
00968
00969
        /* Write profile data...
        if (ctl->prof_basename[0] != '-') {
00971
          sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
00972
          write_prof(filename, ctl, met0, met1, atm, t);
00973
00974
00975
00976
        /* Write station data...
00977
        if (ctl->stat_basename[0] != '-') {
          sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
00978
00979
          write_station(filename, ctl, atm, t);
00980
        1
00981 }
```

# 5.41 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[])

#### 5.41.1 Detailed Description

Create meteorological data files with synthetic wind fields.

Definition in file wind.c.

#### 5.41.2 Function Documentation

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

Definition at line 188 of file wind.c.

### 5.41.2.2 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
00057
        /* Allocate... */
       ALLOC(dataT, float,
00058
00059
              EP * EY * EX);
       ALLOC(dataU, float,
EP * EY * EX);
00060
00061
       ALLOC(dataV, float,
EP * EY * EX);
00062
00063
00064
       ALLOC (dataW, float,
00065
              EP * EY * EX);
00066
00067
        /* Check arguments... */
        if (argc < 3)
00068
00069
          ERRMSG("Give parameters: <ctl> <metbase>");
00070
00071
       /* Read control parameters... */
```

```
read_ctl(argv[1], argc, argv, &ctl);
             read_ctl(argv[1], argc, argv, &ctl);
t0 = scan_ctl(argv[1], argc, argv, "WIND_T0", -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_Z0", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "WIND_Z1", -1, "60", NULL);
00073
00074
00075
00076
00077
             u0 = scan_ctl(argy[1], argc, argv, "WIND_U0", -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_ALPHA", -1, "0.0", NULL);
00079
08000
00081
00082
00083
              /* Check dimensions... */
00084
              if (nx < 1 || nx > EX)
                 ERRMSG("Set 1 <= NX <= MAX!");</pre>
00085
00086
              if (ny < 1 \mid \mid ny > EY)
00087
                ERRMSG("Set 1 <= NY <= MAX!");</pre>
              if (nz < 1 || nz > EP)
00088
                ERRMSG("Set 1 <= NZ <= MAX!");
00089
00090
00091
              /* Get time... */
             jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
t0 = year * 10000. + mon * 100. + day + hour / 24.;
00092
00093
00094
00095
             /* Set filename... */
sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", argv[2], year, mon, day, hour);
00096
00097
00098
              /* Create netCDF file... */
00099
              NC(nc_create(filename, NC_CLOBBER, &ncid));
00100
00101
              /* Create dimensions... */
             /* 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... */
              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));
00108
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, "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
00120
00121
00122
00123
00124
00125
00126
00127
00129
              add_text_attribute(ncid, "U", "units", "m s**-1");
              add_text_attribute(ncid, "V", "long_name", "V velocity");
00130
             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");
00131
00132
00133
00134
00135
              /* End definition... */
00136
             NC(nc_enddef(ncid));
00137
              /* Set coordinates... */
00138
00139
              for (ix = 0; ix < nx; ix++)
                dataLon[ix] = 360.0 / nx * (double) ix;
00140
00141
              for (iy = 0; iy < ny; iy++)
00142
                 dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
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... */
              NC(nc_put_var_double(ncid, timid, &t0));
00147
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));
00151
00152
              /* Create wind fields (Williamson et al., 1992)... */
              for (ix = 0; ix < nx; ix++)
                 for (iy = 0; iy < ny; iy++)</pre>
00154
00155
                     for (iz = 0; iz < nz; iz++) {</pre>
                         idx = (iz * ny + iy) * nx + ix;
00156
                         dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)

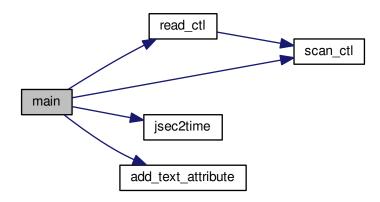
* (cos(dataLat[iy] * M_PI / 180.0)
00157
00158
```

5.42 wind.c 243

```
* 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)));
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));
       NC (nc_put_var_float (ncid, vid, dataV));
00171
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:



# 5.42 wind.c

```
00001 /*
00002
         This file is part of MPTRAC.
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
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) 2013-2018 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
```

```
00027 /* -----
00028
             Functions...
00029
00030
00031 void add text attribute(
00032 int ncid,
00033
            char *varname,
00034
            char *attrname,
00035
            char *text);
00036
00037 /*
00038
            Main...
00039
00040
00041 int main(
00042
            int argc.
00043
            char *argv[]) {
00044
00045
            ctl_t ctl;
00046
00047
            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,
EP * EY * EX);
00060
00061
00062
            ALLOC (dataV, float,
                     EP * EY * EX);
00063
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
00080
            alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);
00081
00082
00083
            /* Check dimensions..
00084
            if (nx < 1 || nx > EX)
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)
00089
              ERRMSG("Set 1 <= NZ <= MAX!");</pre>
00090
            /* Get time... */
00091
00092
            jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
t0 = year * 10000. + mon * 100. + day + hour / 24.;
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
00101
             /* Create dimensions... *
            /* 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... */
            NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));
NC(nc_def_var(ncid, "lev", NC_DOUBLE, 1, &dims[1], &levid));
00108
00109
            NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[1], &levid));
NC(nc_def_var(ncid, "lon", 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));
00110
00111
00112
```

5.42 wind.c 245

```
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], &wid));
00114
00115
00116
         /* 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, "lev", "long_name", "air_pressure");
add_text_attribute(ncid, "lev", "units", "Pa");
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, "V", "units", "m s**-1");
add_text_attribute(ncid, "V", "units", "m s**-1");
add_text_attribute(ncid, "V", "units", "m s**-1");
00117
          /* Set attributes... */
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130
          add_text_attribute(ncid, "V", "units", "m s**-1");
00131
          add_text_attribute(ncid, "W", "long_name", "Vertical velocity");
00132
00133
          add_text_attribute(ncid, "W", "units", "Pa s**-1");
00134
00135
          /* End definition... */
00136
          NC(nc_enddef(ncid));
00137
00138
           /* Set coordinates... */
00139
          for (ix = 0; ix < nx; ix++)
00140
            dataLon[ix] = 360.0 / nx * (double) ix;
00141
          for (iy = 0; iy < ny; iy++)
            dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
00142
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));
00149
          NC(nc_put_var_double(ncid, latid, dataLat));
00150
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++) {
  idx = (iz * ny + iy) * nx + ix;</pre>
00155
00156
                  dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
00157
00158
                                                 * (cos(dataLat[iy] * M_PI / 180.0)
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)));
00162
                  dataV[idx] = (float) (-LIN(0.0, u0, nz - 1.0, u1, iz)
00163
00164
                                                * sin(dataLon[ix] * M_PI / 180.0)
00165
                                                * sin(alpha * M_PI / 180.0));
00166
                }
00167
00168
           /* Write wind data... */
          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... */
00178
          free(dataT);
00179
          free (dataU);
00180
          free (dataV):
00181
          free (dataW):
00182
00183
          return EXIT_SUCCESS;
00184 }
00185
00187
00188 void add_text_attribute(
00189
          int ncid,
00190
          char *varname,
          char *attrname,
00191
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

Index	
add_text_attribute	ctl t, 17
wind.c, 241	csi_ny
atm basename	ctl t, 18
ctl_t, 16	csi nz
atm_dt_out	ctl_t, 17
ctl_t, 16	csi obsfile
atm filter	ctl_t, 17
ctl t, 16	csi obsmin
atm_gpfile	ctl_t, 17
ctl t, 16	csi z0
atm_t, 3	ctl_t, 17
lat, 5	csi z1
lon, 4	ctl_t, 17
np, 4	ctl_t, 5
p, 4	atm_basename, 16
q, 5	atm_dt_out, 16
time, 4	atm_filter, 16
up, 5	atm_gpfile, 16
vp, 5	atm type, 16
wp, 5	balloon, 15
atm_type	csi_basename, 16
ctl_t, 16	csi_dt_out, 17
	csi_lat0, 18
balloon	csi_lat1, 18
ctl_t, 15	csi_lon0, 17
cart 2goo	csi_lon1, 18
cart2geo libtrac.c, 56	csi_modmin, 17
libtrac.h, 135	csi_nx, 17
center.c, 25	csi_ny, 18
main, 25	csi_nz, 17
clim_hno3	csi_obsfile, 17
libtrac.c, 56	csi_obsmin, 17
libtrac.h, 135	csi_z0, 17
clim tropo	csi_z1, 17
libtrac.c, 60	direction, 13
libtrac.h, 138	dt_met, 13
cluster.c, 29	dt_mod, 13
main, 29	ens_basename, 21
conv.c, 35	grid_basename, 18
main, 35	grid_dt_out, 18
csi_basename	grid_gpfile, 18
ctl_t, 16	grid_lat0, 19
csi_dt_out	grid_lat1, 19
ctl_t, 17	grid_lon0, 19
csi_lat0	grid_lon1, 19
ctl_t, 18	grid_nx, 19
csi_lat1	grid_ny, 19
ctl_t, 18	grid_nz, 18
csi_lon0	grid_sparse, 18
ctl_t, 17	grid_z0, 19
csi_lon1	grid_z1, 19
ctl_t, 18	isosurf, 14
csi_modmin	met_dp, 13
ctl_t, 17	met_dx, 13
csi_nx	met_dy, 13

met_geopot, 14	turb_dz_strat, 15
met_np, 14	turb_dz_trop, 15
met_p, 14	turb_mesox, 15
met_sp, 14	turb mesoz, 15
— ·	turb_mesoz, 15
met_stage, 14	day2day
met_sx, 14	day2doy
met_sy, 14	libtrac.c, 61
met_tropo, 14	libtrac.h, 140
molmass, 15	day2doy.c, 36
nq, 10	main, 37
prof_basename, 19	deg2dx
prof_lat0, 20	libtrac.c, 62
prof_lat1, 21	libtrac.h, 141
prof lon0, 20	deg2dy
prof_lon1, 20	libtrac.c, 62
• —	libtrac.h, 141
prof_nx, 20	direction
prof_ny, 20	ctl t, 13
prof_nz, 20	<i>-</i> '
prof_obsfile, 20	dist.c, 38
prof_z0, 20	main, 38
prof_z1, 20	doy2day
psc_h2o, 16	libtrac.c, 62
psc hno3, 16	libtrac.h, 140
qnt_ens, 10	doy2day.c, 45
qnt_format, 10	main, 45
qnt_h2o, 12	dp2dz
• —	libtrac.c, 62
qnt_m, 10	libtrac.h, 141
qnt_name, 10	dt met
qnt_o3, 12	_ ctl_t, 13
qnt_p, 11	dt mod
qnt_ps, 11	ctl_t, 13
qnt_pt, 11	dx2deg
qnt_pv, 12	•
qnt_r, 11	libtrac.c, 63
qnt_rho, 10	libtrac.h, 141
qnt_stat, 13	dy2deg
qnt_t, 11	libtrac.c, 63
gnt theta, 12	libtrac.h, 142
qnt_tice, 12	dz2dp
qnt_tnat, 12	libtrac.c, 63
qnt tsts, 12	libtrac.h, 142
• —	
qnt_u, 11	ens_basename
qnt_unit, 10	ctl_t, 21
qnt_v, 11	extract.c, 46
qnt_vh, 12	main, 47
qnt_vz, 12	
qnt_w, 11	geo2cart
qnt_z, 11	libtrac.c, 63
stat_basename, 21	libtrac.h, 142
stat lat, 21	get_met
stat lon, 21	libtrac.c, 63
stat_r, 21	libtrac.h, 142
t_start, 13	get_met_help
	· ·
t_stop, 13	libtrac.c, 64
tdec_strat, 16	libtrac.h, 143
tdec_trop, 15	grid_basename
turb_dx_strat, 15	ctl_t, 18
turb_dx_trop, 15	grid_dt_out

ctl_t, 18	deg2dy, 62
grid_gpfile	doy2day, 62
ctl_t, 18	dp2dz, 62
grid_lat0	dx2deg, 63
ctl_t, 19	dy2deg, 63
grid_lat1	dz2dp, 63
ctl_t, 19	geo2cart, 63
grid_lon0	get_met, 63
ctl_t, 19	get_met_help, 64
grid_lon1	intpol_met_2d, 65
ctl_t, 19	intpol_met_3d, 65
grid_nx	intpol_met_space, 66
ctl_t, 19	intpol_met_time, 66
grid_ny ctl_t, 19	jsec2time, 67
grid_nz	locate, 68
ctl t, 18	read_atm, 68
grid sparse	read_ctl, 70
ctl_t, 18	read_met, 74
grid_z0	read_met_extrapolate, 76
ctl_t, 19	read_met_geopot, 76 read met help, 78
grid_z1	read_met_ml2pl, 79
ctl_t, 19	read_met_periodic, 79
<u>-</u> -,	read_met_pv, 80
h2o	read_met_sample, 81
met_t, 24	read_met_tropo, 82
	scan_ctl, 84
init.c, 49	time2jsec, 85
main, 49	timer, 86
intpol_met_2d	write_atm, 86
libtrac.c, 65	write_csi, 88
libtrac.h, 144	write_ens, 90
intpol_met_3d	write_grid, 92
libtrac.c, 65	write_prof, 94
libtrac.h, 144	write station, 96
intpol_met_space	libtrac.h, 133
libtrac.c, 66	cart2geo, 135
libtrac.h, 145	clim hno3, 135
intpol_met_time	clim tropo, 138
libtrac.c, 66	day2doy, 140
libtrac.h, 145	deg2dx, 141
isosurf ctl t, 14	deg2dy, 141
Cti_t, 14	doy2day, 140
jsec2time	dp2dz, 141
libtrac.c, 67	dx2deg, 141
libtrac.h, 146	dy2deg, 142
jsec2time.c, 53	dz2dp, 142
main, 53	geo2cart, 142
	get_met, 142
lat	get_met_help, 143
atm_t, 5	intpol_met_2d, 144
met_t, 23	intpol_met_3d, 144
libtrac.c, 54	intpol_met_space, 145
cart2geo, 56	intpol_met_time, 145
clim_hno3, 56	jsec2time, 146
clim_tropo, 60	locate, 147
day2doy, 61	read_atm, 147
deg2dx, 62	read_ctl, 149

	read_met, 153	met	_p
	read_met_extrapolate, 155		ctl_t, 14
	read_met_geopot, 155	met	_prof.c, 193
	read_met_help, 157		main, 194
	read_met_ml2pl, 158	met	_sample.c, 197
	read_met_periodic, 158		main, 198
	read_met_pv, 159	met	_sp
	read_met_sample, 160		ctl_t, 14
	read_met_tropo, 161	met	_stage
	scan_ctl, 163		ctl_t, 14
	time2jsec, 164	met	_sx
	timer, 165		ctl_t, 14
	write_atm, 165	met	_sy
	write_csi, 167		ctl_t, 14
	write_ens, 169	met	_t, 21
	write_grid, 171		h2o, 24
	write_prof, 173		lat, 23
	write station, 175		lon, 23
locat	<del>-</del>		np, 23
	libtrac.c, 68		nx, 22
	libtrac.h, 147		ny, <mark>23</mark>
lon	,		o3, <mark>24</mark>
	atm_t, 4		p, 23
	met t, 23		pl, 24
			ps, 23
mair	1		pt, 23
	center.c, 25		pv, 24
	cluster.c, 29		t, 23
	conv.c, 35		
	day2doy.c, 37		time, 22
	dist.c, 38		u, 24
	doy2day.c, 45		v, 24
	extract.c, 47		w, 24
	init.c, 49		z, 23
	jsec2time.c, 53	met	_tropo
	match.c, 185		ctl_t, 14
	met map.c, 190	met	_zm.c, 201
	<del></del>		main, 202
	met_prof.c, 194	mod	lule_advection
	met_sample.c, 198		trac.c, 215
	met_zm.c, 202	mod	lule_decay
	smago.c, 205		trac.c, 215
	split.c, 209	mod	lule_diffusion_meso
	time2jsec.c, 213		trac.c, 216
	trac.c, 225	mod	lule_diffusion_turb
	wind.c, 241		trac.c, 218
mato	ch.c, 185	mod	lule_isosurf
	main, 185		trac.c, 219
met_	•	mod	lule_meteo
	ctl_t, 13		trac.c, 220
met_		mod	lule_position
	ctl_t, 13		trac.c, 222
met_	_dy	mod	lule_sedi
	ctl_t, 13		trac.c, 223
met_	geopot	moli	mass
	ctl_t, 14		ctl_t, 15
met	 _map.c, 189		∪a_t, 1∪
_	main, 190	np	
met		•	atm_t, 4
	ctl_t, 14		met_t, 23
	_ ·		_ , -

nq	ctl_t, 10
ctl_t, 10	qnt_o3
nx	ctl_t, 12
met t, 22	qnt_p
ny	ctl t, 11
met t, 23	qnt_ps
<u></u>	ctl_t, 11
03	qnt_pt
met_t, 24	
1110t_t, 24	ctl_t, 11
n	qnt_pv
p atm_t, 4	ctl_t, 12
	qnt_r
met_t, 23	ctl_t, 11
pl	qnt_rho
met_t, 24	ctl_t, 10
prof_basename	qnt_stat
ctl_t, 19	ctl_t, 13
prof_lat0	qnt_t
ctl_t, 20	ctl_t, 11
prof_lat1	qnt_theta
ctl_t, 21	ctl_t, 12
prof_lon0	qnt_tice
ctl_t, 20	ctl t, 12
prof lon1	qnt_tnat
ctl_t, 20	
prof_nx	ctl_t, 12
ctl_t, 20	qnt_tsts
prof_ny	ctl_t, 12
ctl_t, 20	qnt_u
	ctl_t, 11
prof_nz	qnt_unit
ctl_t, 20	ctl_t, 10
prof_obsfile	qnt_v
ctl_t, 20	ctl_t, 11
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, 23	ctl_t, 11
psc_h2o	qnt_z
ctl_t, 16	ctl_t, 11
psc_hno3	_ ,
ctl_t, 16	read_atm
pt	libtrac.c, 68
met_t, 23	libtrac.h, 147
pv	read_ctl
met_t, 24	libtrac.c, 70
	libtrac.h, 149
q	read_met
atm_t, 5	libtrac.c, 74
qnt ens	libtrac.h, 153
ctl_t, 10	read_met_extrapolate
qnt_format	libtrac.c, 76
• —	
ctl_t, 10	libtrac.h, 155
qnt_h2o	read_met_geopot
ctl_t, 12	libtrac.c, 76
qnt_m	libtrac.h, 155
ctl_t, 10	read_met_help
qnt_name	libtrac.c, 78

libtrac.h, 157	module_diffusion_turb, 218
read_met_ml2pl	module_isosurf, 219
libtrac.c, 79	module_meteo, 220
libtrac.h, 158	module_position, 222
read_met_periodic	module_sedi, 223
libtrac.c, 79	write_output, 224
libtrac.h, 158	turb_dx_strat
read_met_pv	ctl_t, 15
libtrac.c, 80	turb_dx_trop
libtrac.h, 159	ctl_t, 15
read_met_sample	turb_dz_strat
libtrac.c, 81	ctl_t, 15
libtrac.h, 160	turb_dz_trop
read_met_tropo	ctl_t, 15
libtrac.c, 82	turb_mesox
libtrac.h, 161	ctl_t, 15
u.	turb_mesoz
scan_ctl	ctl_t, 15
libtrac.c, 84	
libtrac.h, 163	u
smago.c, 205	met_t, 24
main, 205	up
split.c, 208	atm_t, 5
main, 209	
stat_basename	V
ctl_t, 21	met_t, 24
stat_lat	vp
ctl_t, 21	atm_t, 5
stat_lon	
ctl_t, 21	W
stat_r	met_t, 24
ctl_t, 21	wind.c, 241
	11 1 1 1 1 1 0 4 4
,	add_text_attribute, 241
t	main, 241
	main, 241 wp
t	main, 241 wp atm_t, 5
t met_t, 23	main, 241 wp atm_t, 5 write_atm
t met_t, 23 t_start	main, 241 wp atm_t, 5 write_atm libtrac.c, 86
t met_t, 23 t_start ctl_t, 13	main, 241 wp atm_t, 5 write_atm libtrac.c, 86 libtrac.h, 165
t met_t, 23 t_start ctl_t, 13 t_stop	main, 241 wp atm_t, 5 write_atm libtrac.c, 86 libtrac.h, 165 write_csi
t met_t, 23 t_start ctl_t, 13 t_stop ctl_t, 13	main, 241  wp atm_t, 5  write_atm libtrac.c, 86 libtrac.h, 165  write_csi libtrac.c, 88
t met_t, 23 t_start ctl_t, 13 t_stop ctl_t, 13 tdec_strat	main, 241  wp atm_t, 5  write_atm libtrac.c, 86 libtrac.h, 165  write_csi libtrac.c, 88 libtrac.h, 167
t met_t, 23 t_start ctl_t, 13 t_stop ctl_t, 13 tdec_strat ctl_t, 16	main, 241  wp atm_t, 5  write_atm libtrac.c, 86 libtrac.h, 165  write_csi libtrac.c, 88 libtrac.h, 167  write_ens
t met_t, 23 t_start ctl_t, 13 t_stop ctl_t, 13 tdec_strat ctl_t, 16 tdec_trop	main, 241  wp atm_t, 5  write_atm libtrac.c, 86 libtrac.h, 165  write_csi libtrac.c, 88 libtrac.h, 167  write_ens libtrac.c, 90
t met_t, 23 t_start ctl_t, 13 t_stop ctl_t, 13 tdec_strat ctl_t, 16 tdec_trop ctl_t, 15	main, 241  wp atm_t, 5  write_atm libtrac.c, 86 libtrac.h, 165  write_csi libtrac.c, 88 libtrac.h, 167  write_ens libtrac.c, 90 libtrac.h, 169
t met_t, 23 t_start ctl_t, 13 t_stop ctl_t, 13 tdec_strat ctl_t, 16 tdec_trop ctl_t, 15 time	main, 241  wp atm_t, 5  write_atm libtrac.c, 86 libtrac.h, 165  write_csi libtrac.c, 88 libtrac.h, 167  write_ens libtrac.c, 90
t met_t, 23 t_start ctl_t, 13 t_stop ctl_t, 13 tdec_strat ctl_t, 16 tdec_trop ctl_t, 15 time atm_t, 4	main, 241  wp atm_t, 5  write_atm libtrac.c, 86 libtrac.h, 165  write_csi libtrac.c, 88 libtrac.h, 167  write_ens libtrac.c, 90 libtrac.h, 169
t met_t, 23 t_start ctl_t, 13 t_stop ctl_t, 13 tdec_strat ctl_t, 16 tdec_trop ctl_t, 15 time atm_t, 4 met_t, 22	main, 241  wp atm_t, 5  write_atm libtrac.c, 86 libtrac.h, 165  write_csi libtrac.c, 88 libtrac.h, 167  write_ens libtrac.c, 90 libtrac.h, 169  write_grid
t met_t, 23 t_start ctl_t, 13 t_stop ctl_t, 13 tdec_strat ctl_t, 16 tdec_trop ctl_t, 15 time atm_t, 4 met_t, 22 time2jsec libtrac.c, 85	main, 241  wp atm_t, 5  write_atm libtrac.c, 86 libtrac.h, 165  write_csi libtrac.c, 88 libtrac.h, 167  write_ens libtrac.c, 90 libtrac.h, 169  write_grid libtrac.c, 92
t met_t, 23 t_start ctl_t, 13 t_stop ctl_t, 13 tdec_strat ctl_t, 16 tdec_trop ctl_t, 15 time atm_t, 4 met_t, 22 time2jsec libtrac.c, 85 libtrac.h, 164	main, 241  wp atm_t, 5  write_atm libtrac.c, 86 libtrac.h, 165  write_csi libtrac.c, 88 libtrac.h, 167  write_ens libtrac.c, 90 libtrac.h, 169  write_grid libtrac.c, 92 libtrac.h, 171
t met_t, 23 t_start ctl_t, 13 t_stop ctl_t, 13 tdec_strat ctl_t, 16 tdec_trop ctl_t, 15 time atm_t, 4 met_t, 22 time2jsec libtrac.c, 85 libtrac.h, 164 time2jsec.c, 212	main, 241  wp atm_t, 5  write_atm libtrac.c, 86 libtrac.h, 165  write_csi libtrac.c, 88 libtrac.h, 167  write_ens libtrac.c, 90 libtrac.h, 169  write_grid libtrac.c, 92 libtrac.h, 171  write_output
t met_t, 23 t_start ctl_t, 13 t_stop ctl_t, 13 tdec_strat ctl_t, 16 tdec_trop ctl_t, 15 time atm_t, 4 met_t, 22 time2jsec libtrac.c, 85 libtrac.h, 164	main, 241  wp atm_t, 5  write_atm libtrac.c, 86 libtrac.h, 165  write_csi libtrac.c, 88 libtrac.h, 167  write_ens libtrac.c, 90 libtrac.h, 169  write_grid libtrac.c, 92 libtrac.h, 171  write_output trac.c, 224
t met_t, 23 t_start ctl_t, 13 t_stop ctl_t, 13 tdec_strat ctl_t, 16 tdec_trop ctl_t, 15 time atm_t, 4 met_t, 22 time2jsec libtrac.c, 85 libtrac.h, 164 time2jsec.c, 212 main, 213 timer	main, 241  wp atm_t, 5  write_atm libtrac.c, 86 libtrac.h, 165  write_csi libtrac.c, 88 libtrac.h, 167  write_ens libtrac.c, 90 libtrac.h, 169  write_grid libtrac.c, 92 libtrac.h, 171  write_output trac.c, 224  write_prof
t met_t, 23 t_start ctl_t, 13 t_stop ctl_t, 13 tdec_strat ctl_t, 16 tdec_trop ctl_t, 15 time atm_t, 4 met_t, 22 time2jsec libtrac.c, 85 libtrac.h, 164 time2jsec.c, 212 main, 213 timer libtrac.c, 86	main, 241  wp atm_t, 5  write_atm libtrac.c, 86 libtrac.h, 165  write_csi libtrac.c, 88 libtrac.h, 167  write_ens libtrac.c, 90 libtrac.h, 169  write_grid libtrac.c, 92 libtrac.h, 171  write_output trac.c, 224  write_prof libtrac.c, 94
t met_t, 23 t_start ctl_t, 13 t_stop ctl_t, 13 tdec_strat ctl_t, 16 tdec_trop ctl_t, 15 time atm_t, 4 met_t, 22 time2jsec libtrac.c, 85 libtrac.h, 164 time2jsec.c, 212 main, 213 timer libtrac.c, 86 libtrac.h, 165	main, 241  wp atm_t, 5  write_atm libtrac.c, 86 libtrac.h, 165  write_csi libtrac.c, 88 libtrac.h, 167  write_ens libtrac.c, 90 libtrac.h, 169  write_grid libtrac.c, 92 libtrac.h, 171  write_output trac.c, 224  write_prof libtrac.c, 94 libtrac.h, 173
t met_t, 23 t_start ctl_t, 13 t_stop ctl_t, 13 tdec_strat ctl_t, 16 tdec_trop ctl_t, 15 time atm_t, 4 met_t, 22 time2jsec libtrac.c, 85 libtrac.h, 164 time2jsec.c, 212 main, 213 timer libtrac.c, 86 libtrac.h, 165 trac.c, 214	main, 241  wp atm_t, 5  write_atm libtrac.c, 86 libtrac.h, 165  write_csi libtrac.c, 88 libtrac.h, 167  write_ens libtrac.c, 90 libtrac.h, 169  write_grid libtrac.c, 92 libtrac.h, 171  write_output trac.c, 224  write_prof libtrac.h, 173  write_station
t met_t, 23 t_start ctl_t, 13 t_stop ctl_t, 13 tdec_strat ctl_t, 16 tdec_trop ctl_t, 15 time atm_t, 4 met_t, 22 time2jsec libtrac.c, 85 libtrac.h, 164 time2jsec.c, 212 main, 213 timer libtrac.c, 86 libtrac.h, 165 trac.c, 214 main, 225	main, 241  wp atm_t, 5  write_atm libtrac.c, 86 libtrac.h, 165  write_csi libtrac.c, 88 libtrac.h, 167  write_ens libtrac.c, 90 libtrac.h, 169  write_grid libtrac.c, 92 libtrac.h, 171  write_output trac.c, 224  write_prof libtrac.c, 94 libtrac.h, 173  write_station libtrac.c, 96
t met_t, 23 t_start ctl_t, 13 t_stop ctl_t, 13 tdec_strat ctl_t, 16 tdec_trop ctl_t, 15 time atm_t, 4 met_t, 22 time2jsec libtrac.c, 85 libtrac.h, 164 time2jsec.c, 212 main, 213 timer libtrac.c, 86 libtrac.h, 165 trac.c, 214 main, 225 module_advection, 215	main, 241  wp atm_t, 5  write_atm libtrac.c, 86 libtrac.h, 165  write_csi libtrac.c, 88 libtrac.h, 167  write_ens libtrac.c, 90 libtrac.h, 169  write_grid libtrac.c, 92 libtrac.h, 171  write_output trac.c, 224  write_prof libtrac.c, 94 libtrac.h, 173  write_station libtrac.c, 96
t met_t, 23 t_start ctl_t, 13 t_stop ctl_t, 13 tdec_strat ctl_t, 16 tdec_trop ctl_t, 15 time atm_t, 4 met_t, 22 time2jsec libtrac.c, 85 libtrac.h, 164 time2jsec.c, 212 main, 213 timer libtrac.c, 86 libtrac.h, 165 trac.c, 214 main, 225	main, 241  wp atm_t, 5  write_atm libtrac.c, 86 libtrac.h, 165  write_csi libtrac.c, 88 libtrac.h, 167  write_ens libtrac.c, 90 libtrac.h, 169  write_grid libtrac.c, 92 libtrac.h, 171  write_output trac.c, 224  write_prof libtrac.h, 173  write_station libtrac.c, 96 libtrac.h, 175