# **MPTRAC**

Generated by Doxygen 1.8.11

ii CONTENTS

# Contents

1	Main	n Page	2
2	Data	Structure Index	2
	2.1	Data Structures	2
3	File	Index	2
	3.1	File List	2
4	Data	Structure Documentation	3
	4.1	atm_t Struct Reference	3
		4.1.1 Detailed Description	4
		4.1.2 Field Documentation	4
	4.2	ctl_t Struct Reference	5
		4.2.1 Detailed Description	8
		4.2.2 Field Documentation	9
	4.3	met_t Struct Reference	18
		4.3.1 Detailed Description	19
		4.3.2 Field Documentation	19
5	File	Documentation	21
	5.1	center.c File Reference	21
		5.1.1 Detailed Description	21
		5.1.2 Function Documentation	22
	5.2	center.c	23
	5.3	dist.c File Reference	25
		5.3.1 Detailed Description	26
		5.3.2 Function Documentation	26
	5.4	dist.c	29
	5.5	extract.c File Reference	32
		5.5.1 Detailed Description	32
		5.5.2 Function Documentation	32

5.6	extract.c	33
5.7	init.c File Reference	34
	5.7.1 Detailed Description	35
	5.7.2 Function Documentation	35
5.8	init.c	36
5.9	jsec2time.c File Reference	38
	5.9.1 Detailed Description	38
	5.9.2 Function Documentation	38
5.10	jsec2time.c	39
5.11	libtrac.c File Reference	39
	5.11.1 Detailed Description	41
	5.11.2 Function Documentation	41
5.12	libtrac.c	70
5.13	libtrac.h File Reference	94
	5.13.1 Detailed Description	95
	5.13.2 Function Documentation	96
5.14	libtrac.h	125
5.15	match.c File Reference	132
	5.15.1 Detailed Description	132
	5.15.2 Function Documentation	132
5.16	match.c	134
5.17	met_map.c File Reference	137
	5.17.1 Detailed Description	137
	5.17.2 Function Documentation	137
5.18	met_map.c	139
5.19	met_prof.c File Reference	141
	5.19.1 Detailed Description	141
	5.19.2 Function Documentation	141
5.20	met_prof.c	143
5.21	met_sample.c File Reference	145

	5.21.2 Function Documentation	145
5.22	met_sample.c	147
5.23	met_zm.c File Reference	148
	5.23.1 Detailed Description	148
	5.23.2 Function Documentation	149
5.24	met_zm.c	
	smago.c File Reference	
0.20	5.25.1 Detailed Description	
	5.25.2 Function Documentation	
F 00		
	smago.c	
5.27	split.c File Reference	
	5.27.1 Detailed Description	
	5.27.2 Function Documentation	157
5.28	split.c	159
5.29	time2jsec.c File Reference	160
	5.29.1 Detailed Description	160
	5.29.2 Function Documentation	161
5.30	time2jsec.c	161
5.31	trac.c File Reference	162
	5.31.1 Detailed Description	162
	5.31.2 Function Documentation	163
5.32	trac.c	177
5.33	wind.c File Reference	189
	5.33.1 Detailed Description	189
	5.33.2 Function Documentation	189
5.34	wind.c	191
dex		195

Index

## 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: http://www.fz-juelich.de/ias/jsc/mptrac

## 2 Data Structure Index

## 2.1 Data Structures

Here are the data structures with brief descriptions:

atm_t	
Atmospheric data	
ctl_t Control parameters	,
met_t	·
Meteorological data	18

## 3 File Index

## 3.1 File List

Here is a list of all files with brief descriptions:

Calculate center of mass of air parcels	21
dist.c Calculate transport deviations of trajectories	25
extract.c Extract single trajectory from atmospheric data files	32
init.c Create atmospheric data file with initial air parcel positions	34
jsec2time.c Convert Julian seconds to date	38
libtrac.c MPTRAC library definitions	39
libtrac.h MPTRAC library declarations	94
match.c Calculate deviations between two trajectories	132

met_map.c Extract global map from meteorological data	137
met_prof.c Extract vertical profile from meteorological data	141
met_sample.c Sample meteorological data at given geolocations	145
met_zm.c Extract zonal mean from meteorological data	148
smago.c Estimate horizontal diffusivity based on Smagorinsky theory	153
split.c Split air parcels into a larger number of parcels	
time2jsec.c Convert date to Julian seconds	160
trac.c  Lagrangian particle dispersion model	162
wind.c Create meteorological data files with synthetic wind fields	189
4 Data Structure Documentation	
4.1 atm_t Struct Reference	
Atmospheric data.	
<pre>#include <libtrac.h></libtrac.h></pre>	
Data Fields	
<ul> <li>int np     Number of air pacels.</li> <li>double time [NP]     Time [s].</li> <li>double p [NP]     Pressure [hPa].</li> <li>double lon [NP]     Longitude [deg].</li> <li>double lat [NP]     Latitude [deg].</li> <li>double q [NQ][NP]     Quantitiy data (for various, user-defined attributes).</li> <li>double up [NP]</li> </ul>	
Zonal wind perturbation [m/s].  • double vp [NP]  Meridional wind perturbation [m/s].  • double wp [NP]	
endering the first 1	

Vertical velocity perturbation [hPa/s].

4

```
4.1.1 Detailed Description
Atmospheric data.
Definition at line 444 of file libtrac.h.
4.1.2 Field Documentation
4.1.2.1 int atm_t::np
Number of air pacels.
Definition at line 447 of file libtrac.h.
4.1.2.2 double atm_t::time[NP]
Time [s].
Definition at line 450 of file libtrac.h.
4.1.2.3 double atm_t::p[NP]
Pressure [hPa].
Definition at line 453 of file libtrac.h.
4.1.2.4 double atm_t::lon[NP]
Longitude [deg].
Definition at line 456 of file libtrac.h.
4.1.2.5 double atm_t::lat[NP]
Latitude [deg].
Definition at line 459 of file libtrac.h.
4.1.2.6 double atm_t::q[NQ][NP]
Quantitiy data (for various, user-defined attributes).
Definition at line 462 of file libtrac.h.
4.1.2.7 double atm_t::up[NP]
Zonal wind perturbation [m/s].
Definition at line 465 of file libtrac.h.
```

```
4.1.2.8 double atm_t::vp[NP]
Meridional wind perturbation [m/s].
Definition at line 468 of file libtrac.h.
4.1.2.9 double atm_t::wp[NP]
Vertical velocity perturbation [hPa/s].
Definition at line 471 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_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\_pv Quantity array index for potential vorticity. · int ant 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\_np Number of target pressure levels. double met\_p [EP] Target pressure levels [hPa]. · int isosurf Isosurface parameter (0=none, 1=pressure, 2=density, 3=theta, 4=balloon). • char balloon [LEN] Balloon position filename. double turb dx trop Horizontal turbulent diffusion coefficient (troposphere) [m^2/s]. double turb\_dx\_strat Horizontal turbulent diffusion coefficient (stratosphere) [m^2/s]. double turb\_dz\_trop Vertical turbulent diffusion coefficient (troposphere)  $[m^2/s]$ . • double turb\_dz\_strat Vertical turbulent diffusion coefficient (stratosphere) [m^2/s]. · double turb meso Scaling factor for mesoscale wind fluctuations. · 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).

• 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<sup>2</sup>].

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 177 of file libtrac.h.

4.2.2 Field Documentation 4.2.2.1 int ctl\_t::nq Number of quantities. Definition at line 180 of file libtrac.h. 4.2.2.2 char ctl\_t::qnt\_name[NQ][LEN] Quantity names. Definition at line 183 of file libtrac.h. 4.2.2.3 char ctl\_t::qnt\_unit[NQ][LEN] Quantity units. Definition at line 186 of file libtrac.h. 4.2.2.4 char ctl\_t::qnt\_format[NQ][LEN] Quantity output format. Definition at line 189 of file libtrac.h. 4.2.2.5 int ctl\_t::qnt\_ens Quantity array index for ensemble IDs. Definition at line 192 of file libtrac.h. 4.2.2.6 int ctl\_t::qnt\_m Quantity array index for mass. Definition at line 195 of file libtrac.h. 4.2.2.7 int ctl\_t::qnt\_rho Quantity array index for particle density. Definition at line 198 of file libtrac.h. 4.2.2.8 int ctl\_t::qnt\_r Quantity array index for particle radius. Definition at line 201 of file libtrac.h.

```
4.2.2.9 int ctl_t::qnt_ps
Quantity array index for surface pressure.
Definition at line 204 of file libtrac.h.
4.2.2.10 int ctl_t::qnt_p
Quantity array index for pressure.
Definition at line 207 of file libtrac.h.
4.2.2.11 int ctl_t::qnt_t
Quantity array index for temperature.
Definition at line 210 of file libtrac.h.
4.2.2.12 int ctl_t::qnt_u
Quantity array index for zonal wind.
Definition at line 213 of file libtrac.h.
4.2.2.13 int ctl_t::qnt_v
Quantity array index for meridional wind.
Definition at line 216 of file libtrac.h.
4.2.2.14 int ctl_t::qnt_w
Quantity array index for vertical velocity.
Definition at line 219 of file libtrac.h.
4.2.2.15 int ctl_t::qnt_h2o
Quantity array index for water vapor vmr.
Definition at line 222 of file libtrac.h.
4.2.2.16 int ctl_t::qnt_o3
Quantity array index for ozone vmr.
Definition at line 225 of file libtrac.h.
4.2.2.17 int ctl_t::qnt_theta
Quantity array index for potential temperature.
```

Definition at line 228 of file libtrac.h.

```
4.2.2.18 int ctl_t::qnt_pv
Quantity array index for potential vorticity.
Definition at line 231 of file libtrac.h.
4.2.2.19 int ctl_t::qnt_tice
Quantity array index for T_ice.
Definition at line 234 of file libtrac.h.
4.2.2.20 int ctl_t::qnt_tsts
Quantity array index for T_STS.
Definition at line 237 of file libtrac.h.
4.2.2.21 int ctl_t::qnt_tnat
Quantity array index for T_NAT.
Definition at line 240 of file libtrac.h.
4.2.2.22 int ctl_t::qnt_stat
Quantity array index for station flag.
Definition at line 243 of file libtrac.h.
4.2.2.23 int ctl_t::direction
Direction flag (1=forward calculation, -1=backward calculation).
Definition at line 246 of file libtrac.h.
4.2.2.24 double ctl_t::t_start
Start time of simulation [s].
Definition at line 249 of file libtrac.h.
4.2.2.25 double ctl_t::t_stop
Stop time of simulation [s].
Definition at line 252 of file libtrac.h.
4.2.2.26 double ctl_t::dt_mod
Time step of simulation [s].
Definition at line 255 of file libtrac.h.
```

```
4.2.2.27 double ctl_t::dt_met
Time step of meteorological data [s].
Definition at line 258 of file libtrac.h.
4.2.2.28 int ctl_t::met_np
Number of target pressure levels.
Definition at line 261 of file libtrac.h.
4.2.2.29 double ctl_t::met_p[EP]
Target pressure levels [hPa].
Definition at line 264 of file libtrac.h.
4.2.2.30 int ctl_t::isosurf
Isosurface parameter (0=none, 1=pressure, 2=density, 3=theta, 4=balloon).
Definition at line 268 of file libtrac.h.
4.2.2.31 char ctl_t::balloon[LEN]
Balloon position filename.
Definition at line 271 of file libtrac.h.
4.2.2.32 double ctl_t::turb_dx_trop
Horizontal turbulent diffusion coefficient (troposphere) [m<sup>2</sup>/s].
Definition at line 274 of file libtrac.h.
4.2.2.33 double ctl_t::turb_dx_strat
Horizontal turbulent diffusion coefficient (stratosphere) [m^2/s].
Definition at line 277 of file libtrac.h.
4.2.2.34 double ctl_t::turb_dz_trop
Vertical turbulent diffusion coefficient (troposphere) [m^2/s].
Definition at line 280 of file libtrac.h.
4.2.2.35 double ctl_t::turb_dz_strat
Vertical turbulent diffusion coefficient (stratosphere) [m^2/s].
Definition at line 283 of file libtrac.h.
```

```
4.2.2.36 double ctl_t::turb_meso
Scaling factor for mesoscale wind fluctuations.
Definition at line 286 of file libtrac.h.
4.2.2.37 double ctl_t::tdec_trop
Life time of particles (troposphere) [s].
Definition at line 289 of file libtrac.h.
4.2.2.38 double ctl_t::tdec_strat
Life time of particles (stratosphere) [s].
Definition at line 292 of file libtrac.h.
4.2.2.39 double ctl_t::psc_h2o
H2O volume mixing ratio for PSC analysis.
Definition at line 295 of file libtrac.h.
4.2.2.40 double ctl_t::psc_hno3
HNO3 volume mixing ratio for PSC analysis.
Definition at line 298 of file libtrac.h.
4.2.2.41 char ctl_t::atm_basename[LEN]
Basename of atmospheric data files.
Definition at line 301 of file libtrac.h.
4.2.2.42 char ctl_t::atm_gpfile[LEN]
Gnuplot file for atmospheric data.
Definition at line 304 of file libtrac.h.
4.2.2.43 double ctl_t::atm_dt_out
Time step for atmospheric data output [s].
Definition at line 307 of file libtrac.h.
4.2.2.44 int ctl_t::atm_filter
Time filter for atmospheric data output (0=no, 1=yes).
Definition at line 310 of file libtrac.h.
```

```
4.2.2.45 char ctl_t::csi_basename[LEN]
Basename of CSI data files.
Definition at line 313 of file libtrac.h.
4.2.2.46 double ctl_t::csi_dt_out
Time step for CSI data output [s].
Definition at line 316 of file libtrac.h.
4.2.2.47 char ctl_t::csi_obsfile[LEN]
Observation data file for CSI analysis.
Definition at line 319 of file libtrac.h.
4.2.2.48 double ctl_t::csi_obsmin
Minimum observation index to trigger detection.
Definition at line 322 of file libtrac.h.
4.2.2.49 double ctl_t::csi_modmin
Minimum column density to trigger detection [kg/m<sup>2</sup>].
Definition at line 325 of file libtrac.h.
4.2.2.50 int ctl_t::csi_nz
Number of altitudes of gridded CSI data.
Definition at line 328 of file libtrac.h.
4.2.2.51 double ctl_t::csi_z0
Lower altitude of gridded CSI data [km].
Definition at line 331 of file libtrac.h.
4.2.2.52 double ctl_t::csi_z1
Upper altitude of gridded CSI data [km].
Definition at line 334 of file libtrac.h.
4.2.2.53 int ctl_t::csi_nx
Number of longitudes of gridded CSI data.
Definition at line 337 of file libtrac.h.
```

```
4.2.2.54 double ctl_t::csi_lon0
Lower longitude of gridded CSI data [deg].
Definition at line 340 of file libtrac.h.
4.2.2.55 double ctl_t::csi_lon1
Upper longitude of gridded CSI data [deg].
Definition at line 343 of file libtrac.h.
4.2.2.56 int ctl_t::csi_ny
Number of latitudes of gridded CSI data.
Definition at line 346 of file libtrac.h.
4.2.2.57 double ctl_t::csi_lat0
Lower latitude of gridded CSI data [deg].
Definition at line 349 of file libtrac.h.
4.2.2.58 double ctl_t::csi_lat1
Upper latitude of gridded CSI data [deg].
Definition at line 352 of file libtrac.h.
4.2.2.59 char ctl_t::grid_basename[LEN]
Basename of grid data files.
Definition at line 355 of file libtrac.h.
4.2.2.60 char ctl_t::grid_gpfile[LEN]
Gnuplot file for gridded data.
Definition at line 358 of file libtrac.h.
4.2.2.61 double ctl_t::grid_dt_out
Time step for gridded data output [s].
Definition at line 361 of file libtrac.h.
4.2.2.62 int ctl_t::grid_sparse
Sparse output in grid data files (0=no, 1=yes).
Definition at line 364 of file libtrac.h.
```

```
4.2.2.63 int ctl_t::grid_nz
Number of altitudes of gridded data.
Definition at line 367 of file libtrac.h.
4.2.2.64 double ctl_t::grid_z0
Lower altitude of gridded data [km].
Definition at line 370 of file libtrac.h.
4.2.2.65 double ctl_t::grid_z1
Upper altitude of gridded data [km].
Definition at line 373 of file libtrac.h.
4.2.2.66 int ctl_t::grid_nx
Number of longitudes of gridded data.
Definition at line 376 of file libtrac.h.
4.2.2.67 double ctl_t::grid_lon0
Lower longitude of gridded data [deg].
Definition at line 379 of file libtrac.h.
4.2.2.68 double ctl_t::grid_lon1
Upper longitude of gridded data [deg].
Definition at line 382 of file libtrac.h.
4.2.2.69 int ctl_t::grid_ny
Number of latitudes of gridded data.
Definition at line 385 of file libtrac.h.
4.2.2.70 double ctl_t::grid_lat0
Lower latitude of gridded data [deg].
Definition at line 388 of file libtrac.h.
4.2.2.71 double ctl_t::grid_lat1
Upper latitude of gridded data [deg].
Definition at line 391 of file libtrac.h.
```

4.2.2.72 char ctl\_t::prof\_basename[LEN] Basename for profile output file. Definition at line 394 of file libtrac.h. 4.2.2.73 char ctl\_t::prof\_obsfile[LEN] Observation data file for profile output. Definition at line 397 of file libtrac.h. 4.2.2.74 int ctl\_t::prof\_nz Number of altitudes of gridded profile data. Definition at line 400 of file libtrac.h. 4.2.2.75 double ctl\_t::prof\_z0 Lower altitude of gridded profile data [km]. Definition at line 403 of file libtrac.h. 4.2.2.76 double ctl\_t::prof\_z1 Upper altitude of gridded profile data [km]. Definition at line 406 of file libtrac.h. 4.2.2.77 int ctl\_t::prof\_nx Number of longitudes of gridded profile data. Definition at line 409 of file libtrac.h. 4.2.2.78 double ctl\_t::prof\_lon0 Lower longitude of gridded profile data [deg]. Definition at line 412 of file libtrac.h. 4.2.2.79 double ctl\_t::prof\_lon1 Upper longitude of gridded profile data [deg]. Definition at line 415 of file libtrac.h. 4.2.2.80 int ctl\_t::prof\_ny Number of latitudes of gridded profile data.

Definition at line 418 of file libtrac.h.

```
4.2.2.81 double ctl_t::prof_lat0
Lower latitude of gridded profile data [deg].
Definition at line 421 of file libtrac.h.
4.2.2.82 double ctl_t::prof_lat1
Upper latitude of gridded profile data [deg].
Definition at line 424 of file libtrac.h.
4.2.2.83 char ctl_t::ens_basename[LEN]
Basename of ensemble data file.
Definition at line 427 of file libtrac.h.
4.2.2.84 char ctl_t::stat_basename[LEN]
Basename of station data file.
Definition at line 430 of file libtrac.h.
4.2.2.85 double ctl_t::stat_lon
Longitude of station [deg].
Definition at line 433 of file libtrac.h.
4.2.2.86 double ctl_t::stat_lat
Latitude of station [deg].
Definition at line 436 of file libtrac.h.
4.2.2.87 double ctl_t::stat_r
Search radius around station [km].
Definition at line 439 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].

float pl [EX][EY][EP]

Pressure on model levels [hPa].

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 h2o [EX][EY][EP]

Water vapor volume mixing ratio [1].

float o3 [EX][EY][EP]

Ozone volume mixing ratio [1].

#### 4.3.1 Detailed Description

Meteorological data.

Definition at line 476 of file libtrac.h.

## 4.3.2 Field Documentation

## 4.3.2.1 double met\_t::time

Time [s].

Definition at line 479 of file libtrac.h.

```
4.3.2.2 int met_t::nx
Number of longitudes.
Definition at line 482 of file libtrac.h.
4.3.2.3 int met_t::ny
Number of latitudes.
Definition at line 485 of file libtrac.h.
4.3.2.4 int met_t::np
Number of pressure levels.
Definition at line 488 of file libtrac.h.
4.3.2.5 double met_t::lon[EX]
Longitude [deg].
Definition at line 491 of file libtrac.h.
4.3.2.6 double met_t::lat[EY]
Latitude [deg].
Definition at line 494 of file libtrac.h.
4.3.2.7 double met_t::p[EP]
Pressure [hPa].
Definition at line 497 of file libtrac.h.
4.3.2.8 double met_t::ps[EX][EY]
Surface pressure [hPa].
Definition at line 500 of file libtrac.h.
4.3.2.9 float met_t::pl[EX][EY][EP]
Pressure on model levels [hPa].
Definition at line 503 of file libtrac.h.
4.3.2.10 float met_t::t[EX][EY][EP]
Temperature [K].
Definition at line 506 of file libtrac.h.
```

5 File Documentation 21

```
4.3.2.11 float met_t::u[EX][EY][EP]
Zonal wind [m/s].
Definition at line 509 of file libtrac.h.
4.3.2.12 float met_t::v[EX][EY][EP]
Meridional wind [m/s].
Definition at line 512 of file libtrac.h.
4.3.2.13 float met_t::w[EX][EY][EP]
Vertical wind [hPa/s].
Definition at line 515 of file libtrac.h.
4.3.2.14 float met_t::h2o[EX][EY][EP]
Water vapor volume mixing ratio [1].
Definition at line 518 of file libtrac.h.
4.3.2.15 float met_t::o3[EX][EY][EP]
Ozone volume mixing ratio [1].
Definition at line 521 of file libtrac.h.
The documentation for this struct was generated from the following file:
    · libtrac.h
   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
```

Generated by Doxygen

Definition in file center.c.

Calculate center of mass of air parcels.

#### 5.1.2 Function Documentation

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

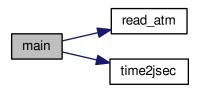
Definition at line 28 of file center.c.

```
00030
00031
00032
        ctl_t ctl;
00033
00034
        atm t *atm;
00035
00036
        FILE *out;
00037
00038
        char *name, *year, *mon, *day, *hour, *min;
00039
00040
        double latm, lats, lonm, lons, t, zm, zs;
00041
00042
        int i, f, ip;
00043
00044
        /* Allocate... */
00045
        ALLOC(atm, atm_t, 1);
00046
00047
        /* Check arguments... */
00048
        if (argc < 3)
00049
          ERRMSG("Give parameters: <outfile> <atml> [<atm2> ...]");
00050
00051
        /\star Write info... \star/
00052
        printf("Write center of mass data: %s\n", argv[1]);
00053
00054
        /\star Create output file...
        if (!(out = fopen(argv[1], "w")))
00055
00056
          ERRMSG("Cannot create file!");
00057
        /* Write header... */
00058
        fprintf(out,
00059
                 "# $1
00060
                       = time [s]\n"
00061
                 "# $2 = altitude (mean) [km]\n"
00062
                 "# $3
                       = altitude (sigma) [km]\n"
00063
                 "# $4 = altitude (minimum) [km]\n"
                 "# $5 = altitude (10%% percentile) [km]\n"
00064
                 "# $6 = altitude (1st quarter) [km]\n"
00065
                 "# $7 = altitude (median) [km]\n"
00066
                 "# $8 = altitude (3rd quarter) [km]\n"
00067
00068
                 "# $9 = altitude (90%% percentile) [km]\n"
                 "# $10 = altitude (maximum) [km] \n");
00069
        fprintf(out, "# $11 = longitude (mean) [deg]\n"
00070
00071
00072
                 "# $12 = longitude (sigma) [deg]\n"
                 "# $13 = longitude (minimum) [deg]\n"
00073
00074
                 "# $14 = longitude (10%% percentile) [deg]\n"
00075
                 "# $15 = longitude (1st quarter) [deg] \n"
                 "# $16 = longitude (median) [deg]\n"
00076
                 "# $17 = longitude (3rd quarter) [deg]\n"
"# $18 = longitude (90%% percentile) [deg]\n"
00077
00078
00079
                 "# $19 = longitude (maximum) [deg]\n");
08000
        fprintf(out,
                 "# $20 = latitude (mean) [deg]\n"
"# $21 = latitude (sigma) [deg]\n"
00081
00082
                 "# $22 = latitude (minimum) [deg]\n"
00083
00084
                 "# $23 = latitude (10%% percentile) [deg]\n"
                 "# $24 = latitude (1st quarter) [deg]\n'
00085
                 "# $25 = latitude (median) [deg] \n"
00086
00087
                 "# $26 = latitude (3rd quarter) [deg]\n"
                 "# $27 = latitude (90%% percentile) [deg]\n"
00088
                 "# $28 = latitude (maximum) [deg] \n\n");
00089
00090
00091
        /* Loop over files... */
        for (f = 2; f < argc; f++) {
00092
00093
00094
          /* Read atmopheric data... */
00095
          read_atm(argv[f], &ctl, atm);
00096
00097
          /* Initialize... */
00098
          zm = zs = 0;
          lonm = lons = 0;
00099
00100
          latm = lats = 0;
00101
          /* Calculate mean and standard deviation... */
for (ip = 0; ip < atm->np; ip++) {
00102
00103
00104
           zm += Z(atm->p[ip]) / atm->np;
00105
            lonm += atm->lon[ip] / atm->np;
```

5.2 center.c 23

```
latm += atm->lat[ip] / atm->np;
                   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;
00108
00109
00110
00111
                 /* Normalize... */
00112
00113
                zs = sqrt(zs - gsl_pow_2(zm));
                lons = sqrt(lons - gsl_pow_2(lonm));
lats = sqrt(lats - gsl_pow_2(latm));
00114
00115
00116
                /* 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);
00117
00118
00119
00120
00121
                /* Get date from filename... */
for (i = (int) strlen(argv[f]) - 1; argv[f][i] != '/' || i == 0; i--);
name = strtok(&(argv[f][i]), "_");
00122
00123
00124
                year = strtok(NULL, "_");
mon = strtok(NULL, "_");
day = strtok(NULL, "_");
00125
00126
00127
                hour = strtok(NULL, "_");
name = strtok(NULL, "_");
min = strtok(name, ".");
00128
00129
                                                               /* TODO: Why another "name" here? */
00130
00131
                time2jsec(atoi(year), atoi(mon), atoi(day), atoi(hour), atoi(min), 0, 0,
00132
00133
                00134
00135
00136
00137
                             Z(atm->p[atm->np - atm->np / 10]),
Z(atm->p[atm->np - atm->np / 4]),
00138
00139
                             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],
00140
00141
00142
00143
                             atm->lon[atm->np - atm->np / 4],
atm->lon[atm->np - atm->np / 4],
atm->lon[atm->np - atm->np / 10],
atm->lon[atm->np - 1],
00144
00145
00146
                             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
            /* Close file... */
00154
            fclose(out);
00155
00156
             /* Free... */
00157
            free(atm);
00158
00159
             return EXIT_SUCCESS;
00160 }
```

Here is the call graph for this function:



#### 5.2 center.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
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
        Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026 #include <gsl/gsl_sort.h>
00027
00028 int main(
00029
       int argc,
       char *argv[]) {
00030
00031
00032
       ctl_t ctl;
00033
00034
       atm_t *atm;
00035
00036
       FILE *out;
00037
00038
        char *name, *year, *mon, *day, *hour, *min;
00039
00040
        double latm, lats, lonm, lons, t, zm, zs;
00041
00042
        int i, f, ip;
00043
00044
        /* Allocate... */
00045
        ALLOC(atm, atm_t, 1);
00046
00047
        /* Check arguments... */
00048
        if (argc < 3)
00049
         ERRMSG("Give parameters: <outfile> <atm1> [<atm2> ...]");
00050
00051
        /* Write info... */
00052
        printf("Write center of mass data: %s\n", argv[1]);
00053
        /* Create output file... */
if (!(out = fopen(argv[1], "w")))
00054
00055
00056
          ERRMSG("Cannot create file!");
00057
00058
        /* Write header... */
00059
        fprintf(out,
00060
                "# $1
                       = time [s]\n"
                "# $2 = altitude (mean) [km]\n"
00061
00062
                 "# $3 = altitude (sigma) [km]\n"
                "# $4 = altitude (minimum) [km] \n"
00063
00064
                "# $5 = altitude (10%% percentile) [km]\n"
00065
                "# $6 = altitude (1st quarter) [km]\n"
                 "# $7 = altitude (median) [km]\n"
00066
                "# $8 = altitude (3rd quarter) [km]\n"
"# $9 = altitude (90%% percentile) [km]\n"
00067
00068
                "# $10 = altitude (maximum) [km]\n");
00069
00070
        fprintf(out,
00071
                 "# $11 = longitude (mean) [deg] n"
                "# $12 = longitude (sigma) [deg] \n"
00072
                 "# $13 = longitude (minimum) [deg] \n"
00073
00074
                "# $14 = longitude (10%% percentile) [deg]\n"
00075
                "# $15 = longitude (1st quarter) [deg]\n"
                "# $16 = longitude (median) [deg]\n"
00076
00077
                "# $17 = longitude (3rd quarter) [deg]\n"
                "# $18 = longitude (90%% percentile) [deg]\n"
00078
                "# $19 = longitude (maximum) [deg]\n");
00079
00080
        fprintf(out,
00081
                 "# $20 = latitude (mean) [deg] \n
                "# $21 = latitude (sigma) [deg]\n"
00082
00083
                "# $22 = latitude (minimum) [deg] n"
00084
                "# $23 = latitude (10%% percentile) [deg]\n"
                 "# $24 = latitude (1st quarter) [deg] n
00085
                "# $25 = latitude (median) [deg] \n"
00086
                "# $26 = latitude (3rd quarter) [deg]\n"
00087
                "# $27 = latitude (90%% percentile) [deg]\n"
00088
00089
                "# $28 = latitude (maximum) [deg] \n\n");
00090
       /* Loop over files... */
for (f = 2; f < argc; f++) {
00091
00092
00093
```

5.3 dist.c File Reference 25

```
/* Read atmopheric data... */
00095
                read_atm(argv[f], &ctl, atm);
00096
                /* Initialize... */
00097
00098
               zm = zs = 0;

lonm = lons = 0;
00099
                latm = lats = 0;
00100
00101
00102
                /* Calculate mean and standard deviation... */
               for (ip = 0; ip < atm->np; ip++) {
  zm += Z(atm->p[ip]) / atm->np;
  lonm += atm->lon[ip] / atm->np;
00103
00104
00105
                   latm += atm->lat[ip] / atm->np;
00106
00107
                   zs += gsl_pow_2(Z(atm->p[ip])) / atm->np;
00108
                   lons += gsl_pow_2(atm->lon[ip]) / atm->np;
                   lats += gsl_pow_2(atm->lat[ip]) / atm->np;
00109
00110
00111
00112
               /* Normalize... */
00113
                zs = sqrt(zs - gsl_pow_2(zm));
               lons = sqrt(lons - gsl_pow_2(lonm));
lats = sqrt(lats - gsl_pow_2(latm));
00114
00115
00116
               /* Sort arrays... */
gsl_sort(atm->p, 1, (size_t) atm->np);
gsl_sort(atm->lon, 1, (size_t) atm->np);

// Sort arrays... */
gsl_sort(atm->lon, 1, (size_t) atm->np);
00117
00118
00119
00120
                gsl_sort(atm->lat, 1, (size_t) atm->np);
00121
               /* Get date from filename... */
for (i = (int) strlen(argv[f]) - 1; argv[f][i] != '/' || i == 0; i--);
name = strtok(&(argv[f][i]), "_");
year = strtok(NULL, "_");
mon = strtok(NULL, "_");
day = strtok(NULL, "_");
00122
00123
00124
00125
00126
00127
               hour = strtok(NULL, "_");
name = strtok(NULL, "_");
min = strtok(name, ".");
00128
                                                             /* TODO: Why another "name" here? */
00129
00130
00131
                time2jsec(atoi(year), atoi(mon), atoi(day), atoi(hour), atoi(min), 0, 0,
00132
                                &t);
00133
               00134
00135
00136
00137
                           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]),
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],
atm->lon[atm->np - atm->np / 4],
atm->lon[atm->np - atm->np / 10],
atm->lon[atm->np - atm->np / 10],
atm->lon[atm->np - 1],
00138
00139
00140
00141
00142
00143
00144
00145
                            atm \rightarrow lon[atm \rightarrow np - 1],
00146
00147
                            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]);
00148
00149
00151
00152
00153
            /* Close file... */
00154
            fclose(out);
00155
00156
            /* Free... */
00157
            free(atm);
00158
00159
            return EXIT_SUCCESS;
00160 }
```

## 5.3 dist.c File Reference

Calculate transport deviations of trajectories.

## **Functions**

• int main (int argc, char \*argv[])

## 5.3.1 Detailed Description

Calculate transport deviations of trajectories.

Definition in file dist.c.

#### 5.3.2 Function Documentation

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

Definition at line 28 of file dist.c.

```
00030
                         {
00032
        ctl_t ctl;
00033
00034
        atm_t *atm1, *atm2;
00035
00036
        FILE *out;
00037
00038
        char *name, *year, *mon, *day, *hour, *min;
00039
        double aux, x0[3], x1[3], x2[3], *lon1, *lat1, *p1, *lh1, *lv1,
  *lon2, *lat2, *p2, *lh2, *lv2, ahtd, avtd, ahtd2, avtd2,
  rhtd, rvtd, rhtd2, rvtd2, t, *dh, *dv;
00040
00041
00042
00043
00044
        int f, i, ip, iph, ipv;
00045
00046
        /* Allocate... */
00047
        ALLOC(atm1, atm_t, 1);
        ALLOC(atm2, atm_t, 1);
ALLOC(lon1, double,
00048
00049
00050
              NP);
00051
        ALLOC(lat1, double,
00052
              NP);
        ALLOC(p1, double,
00053
00054
               NP);
00055
        ALLOC(lh1, double,
00056
               NP);
00057
        ALLOC(lv1, double,
00058
              NP);
00059
        ALLOC(lon2, double,
00060
              NP);
00061
        ALLOC(lat2, double,
              NP);
00062
00063
        ALLOC(p2, double,
00064
               NP);
        ALLOC(lh2, double,
00065
00066
              NP);
        ALLOC(1v2, double,
00067
00068
               NP);
00069
        ALLOC(dh, double,
00070
              NP);
        ALLOC(dv, double,
00071
00072
              NP);
00073
00074
        /* Check arguments... */
00075
        if (argc < 4)
00076
         ERRMSG
00077
             ("Give parameters: <outfile> <atm1a> <atm1b> [<atm2a> <atm2b> ...]");
00078
00079
        /* Write info... */
        printf("Write transport deviations: %s\n", argv[1]);
08000
00081
00082
        /* Create output file... */
        if (!(out = fopen(argv[1], "w")))
    ERRMSG("Cannot create file!");
00083
00084
00085
00086
        /* Write header... */
00087
        fprintf(out,
00088
                  "# $1
                         = time [s]\n"
00089
                  "# $2
                         = AHTD (mean) [km]\n"
                  "# $3 = AHTD (sigma) [km] \n"
00090
                  "# $4 = AHTD (minimum) [km]\n"
00091
                  "# $5 = AHTD (10%% percentile) [km]\n"
00092
00093
                  "# $6 = AHTD (1st quartile) [km]\n'
00094
                  "# $7 = AHTD (median) [km]\n"
```

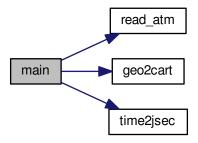
27

```
"# $8 = AHTD (3rd quartile) [km] n"
                   "# $9 = AHTD (90%% percentile) [km]\n"
00096
                   "# $10 = AHTD (maximum) [km] \n"
00097
                   "# $11 = AHTD (maximum trajectory index)\n"  
"# $12 = RHTD (mean) [%^{n}\n"  
# $13 = RHTD (sigma) [%^{n}\n");
00098
00099
00100
         fprintf(out,
                   "# $14 = AVTD (mean) [km] \n"
00102
                   "# $15 = AVTD (sigma) [km] n"
00103
                   "# $16 = AVTD (minimum) [km]n"
                   "# $17 = AVTD (10%% percentile) [km]\n"
00104
                   "# $18 = AVTD (1st quartile) [km]\n"
00105
                   "# $19 = AVTD (median) [km] \n"
00106
                  "# $20 = AVTD (3rd quartile) [km]\n"
"# $21 = AVTD (90%% percentile) [km]\n"
00107
00108
00109
                   "# $22 = AVTD (maximum) [km] \n"
                  "# $23 = AVTD (maximum trajectory index)\n" "# $24 = RVTD (mean) [%%]\n" "# $25 = RVTD (sigma) [%%]\n\n");
00110
00111
00112
00113
         /* Loop over file pairs... */
00114
         for (f = 2; f < argc; f += 2) {</pre>
00115
00116
            /* Read atmopheric data... */
           read_atm(argv[f], &ctl, atm1);
read_atm(argv[f + 1], &ctl, atm2);
00117
00118
00119
00120
            /* Check if structs match... */
00121
           if (atm1->np != atm2->np)
00122
             ERRMSG("Different numbers of parcels!");
            for (ip = 0; ip < atm1->np; ip++)
  if (atm1->time[ip] != atm2->time[ip])
00123
00124
00125
                ERRMSG("Times do not match!");
00126
00127
            /* Init... */
           ahtd = ahtd2 = 0;
avtd = avtd2 = 0;
00128
00129
           rhtd = rhtd2 = 0;
00130
00131
           rvtd = rvtd2 = 0;
00132
00133
            /* Loop over air parcels... */
00134
           for (ip = 0; ip < atm1->np; ip++) {
00135
              /* Get Cartesian coordinates... */
geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00136
00137
00138
              geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00139
00140
              /* Calculate absolute transport deviations... */
              dh[ip] = DIST(x1, x2);
ahtd += dh[ip];
00141
00142
              ahtd2 += gsl_pow_2(dh[ip]);
00143
00144
00145
             dv[ip] = fabs(Z(atm1->p[ip]) - Z(atm2->p[ip]));
00146
              avtd += dv[ip];
00147
              avtd2 += gsl_pow_2(dv[ip]);
00148
00149
              /* Calculate relative transport deviations... */
00150
              if (f > 2) {
00152
                /* Get trajectory lengths... */
00153
                geo2cart(0, lon1[ip], lat1[ip], x0);
                lh1[ip] += DIST(x0, x1);
lv1[ip] += fabs(Z(p1[ip]) - Z(atm1->p[ip]));
00154
00155
00156
00157
                geo2cart(0, lon2[ip], lat2[ip], x0);
                lh2[ip] += DIST(x0, x2);
lv2[ip] += fabs(Z(p2[ip]) - Z(atm2->p[ip]));
00158
00159
00160
00161
                /\star Get relative transport devations... \star/
                if (lh1[ip] + lh2[ip] > 0) {
   aux = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00162
00163
                   rhtd += aux;
00164
00165
                   rhtd2 += gsl_pow_2(aux);
00166
00167
                if (lv1[ip] + lv2[ip] > 0) {
00168
                  aux =
                     200. * fabs(Z(atm1->p[ip]) - Z(atm2->p[ip])) / (lv1[ip] +
00169
00170
                                                                               lv2[ip]);
00171
                   rvtd += aux;
00172
                   rvtd2 += gsl_pow_2(aux);
00173
00174
00175
00176
              /* Save positions of air parcels... */
              lon1[ip] = atm1->lon[ip];
lat1[ip] = atm1->lat[ip];
00177
00178
00179
              p1[ip] = atm1->p[ip];
00180
00181
              lon2[ip] = atm2->lon[ip];
```

```
00182
               lat2[ip] = atm2->lat[ip];
00183
               p2[ip] = atm2->p[ip];
00184
00185
00186
             /\star Get indices of trajectories with maximum errors... \star/
             iph = (int) gsl_stats_max_index(dh, 1, (size_t) atml->np);
ipv = (int) gsl_stats_max_index(dv, 1, (size_t) atml->np);
00187
00188
00189
             /\star Sort distances to calculate percentiles... \star/
00190
             gsl_sort(dh, 1, (size_t) atml->np);
gsl_sort(dv, 1, (size_t) atml->np);
00191
00192
00193
00194
             /* Get date from filename... */
             for (i = (int) strlen(argv[f]) - 1; argv[f][i] != '/' || i == 0; i--);
name = strtok(&(argv[f][i]), "_");
00195
00196
             year = strtok(NULL, "_");
mon = strtok(NULL, "_");
day = strtok(NULL, "_");
00197
00198
00199
             hour = strtok(NULL, "_");
name = strtok(NULL, "_");
min = strtok(name, ".");
00200
00201
                                                  /* TODO: Why another "name" here? */
00202
00203
             time2jsec(atoi(year), atoi(mon), atoi(day), atoi(hour), atoi(min), 0, 0,
00204
                          &t);
00205
00206
             /* Write output... */
             00208
                        " %g %g %g %g %g %g %g %g %d %g %g\n", t,
00209
                        ahtd / atm1->np,
00210
                        sqrt(ahtd2 / atm1->np - gsl_pow_2(ahtd / atm1->np)),
                       dh[0], dh[atml->np / 10], dh[atml->np / 4], dh[atml->np / 2], dh[atml->np - atml->np / 4], dh[atml->np - atml->np / 10], dh[atml->np - atml->np / 10], dh[atml->np - 1], iph, rhtd / atml->np,
00211
00212
00213
00214
                        sqrt(rhtd2 / atm1->np - gsl_pow_2(rhtd / atm1->np)),
00215
                        avtd / atm1->np,
                       dvtd / dtml >np / gsl_pow_2(avtd / atml->np)),
dv[0], dv[atml->np / 10], dv[atml->np / 4], dv[atml->np / 2],
dv[atml->np - atml->np / 4], dv[atml->np - atml->np / 10],
dv[atml->np - 1], ipv, rvtd / atml->np,
00216
00217
00218
00220
                       sqrt(rvtd2 / atm1->np - gsl_pow_2(rvtd / atm1->np)));
00221
00222
          /* Close file... */
00223
00224
          fclose(out):
00225
00226
00227
          free(atm1);
00228
          free(atm2);
00229
          free(lon1);
00230
          free(lat1);
00231
          free(p1);
00232
          free(lh1);
00233
          free(lv1);
00234
          free(lon2);
00235
          free(lat2);
00236
          free (p2);
00237
          free(lh2);
00238
          free(lv2);
00239
          free (dh);
00240
          free(dv);
00241
00242
          return EXIT SUCCESS;
00243 }
```

5.4 dist.c 29

Here is the call graph for this function:



## 5.4 dist.c

```
00001 /*
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,
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
        Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026 #include <gsl/gsl_sort.h>
00027
00028 int main(
00029
        int argc,
00030
        char *argv[]) {
00031
00032
        ctl t ctl;
00033
00034
        atm_t *atm1, *atm2;
00035
00036
        FILE *out;
00037
00038
        char *name, *year, *mon, *day, *hour, *min;
00039
00040
        double aux, x0[3], x1[3], x2[3], *lon1, *lat1, *p1, *lh1, *lv1,
          *lon2, *lat2, *p2, *lh2, *lv2, ahtd, avtd, ahtd2, avtd2, rhtd, rvtd, rhtd2, rvtd2, t, *dh, *dv;
00041
00042
00043
00044
        int f, i, ip, iph, ipv;
00045
00046
         /* Allocate... */
00047
        ALLOC(atm1, atm_t, 1);
00048
        ALLOC(atm2, atm_t, 1);
00049
        ALLOC(lon1, double,
00050
               NP);
00051
        ALLOC(lat1, double,
00052
               NP);
00053
        ALLOC(p1, double,
00054
               NP);
        ALLOC(lh1, double,
00055
00056
               NP);
        ALLOC(lv1, double,
00057
00058
               NP);
00059
        ALLOC(lon2, double,
```

```
NP);
00061
        ALLOC(lat2, double,
00062
              NP);
        ALLOC(p2, double,
00063
00064
              NP);
        ALLOC(lh2, double,
00065
00066
               NP);
00067
        ALLOC(1v2, double,
00068
               NP);
        ALLOC(dh, double,
00069
00070
              NP);
00071
        ALLOC(dv. double.
00072
               NP);
00073
00074
        /* Check arguments... ∗/
00075
        if (argc < 4)</pre>
00076
          ERRMSG
00077
             ("Give parameters: <outfile> <atmla> <atmlb> [<atm2a> <atm2b> ...]");
00078
00079
         /* Write info... */
00080
        printf("Write transport deviations: %s\n", argv[1]);
00081
00082
        /* Create output file... */
        if (!(out = fopen(argv[1], "w")))
00083
00084
          ERRMSG("Cannot create file!");
00085
00086
        /* Write header... */
00087
        fprintf(out,
00088
                 "# $1
                         = time [s]\n"
                  "# $2 = AHTD (mean) [km]\n"
00089
00090
                  "# $3 = AHTD (sigma) [km] \n"
00091
                  "# $4 = AHTD (minimum) [km] \n"
00092
                  "# $5 = AHTD (10%% percentile) [km]\n"
00093
                  "# $6 = AHTD (1st quartile) [km]\n"
                  "# $7 = AHTD (median) [km]\n"
00094
                  "# $8 = AHTD (3rd quartile) [km]\n"
00095
                 "# $9 = AHTD (90%% percentile) [km]\n"
"# $10 = AHTD (maximum) [km]\n"
00096
                 "# $11 = AHTD (maximum trajectory index)\n"
"# $12 = RHTD (mean) [%%]\n" "# $13 = RHTD (sigma) [%%]\n");
00098
00099
        fprintf(out,
    "# $14 = AVTD (mean) [km]\n"
00100
00101
                  "# $15 = AVTD (sigma) [km] \n"
00102
                  "# $16 = AVTD (minimum) [km]\n"
00103
                  "# $17 = AVTD (10%% percentile) [km]\n"
00104
00105
                  "# $18 = AVTD (1st quartile) [km]\n"
00106
                  "# $19 = AVTD \pmod{n} \lceil km \rceil \n"
                  "# $20 = AVTD (3rd quartile) [km]\n"
00107
                  "# $21 = AVTD (90%% percentile) [km]\n"
00108
                  "# $22 = AVTD (maximum) [km]\n"
00109
                  "# $23 = AVTD (maximum trajectory index)\n"
"# $24 = RVTD (mean) [%%]\n" "# $25 = RVTD (sigma) [%%]\n\n");
00110
00111
00112
        /* Loop over file pairs... */
for (f = 2; f < argc; f += 2) {</pre>
00113
00114
00115
           /* Read atmopheric data... */
          read_atm(argv[f], &ctl, atml);
read_atm(argv[f + 1], &ctl, atm2);
00117
00118
00119
00120
           /* Check if structs match... */
          if (atm1->np != atm2->np)
00121
00122
            ERRMSG("Different numbers of parcels!");
00123
           for (ip = 0; ip < atm1->np; ip++)
00124
            if (atm1->time[ip] != atm2->time[ip])
               ERRMSG("Times do not match!");
00125
00126
00127
           /* Init... */
          ahtd = ahtd2 = 0;
00128
           avtd = avtd2 = 0;
00129
00130
           rhtd = rhtd2 = 0;
00131
           rvtd = rvtd2 = 0;
00132
00133
           /* Loop over air parcels... */
00134
           for (ip = 0; ip < atm1->np; ip++) {
00135
00136
             /* Get Cartesian coordinates... */
             geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00137
00138
00139
00140
             /* Calculate absolute transport deviations... */
             dh[ip] = DIST(x1, x2);
ahtd += dh[ip];
00141
00142
00143
             ahtd2 += gsl_pow_2(dh[ip]);
00144
             dv[ip] = fabs(Z(atm1->p[ip]) - Z(atm2->p[ip]));
00145
             avtd += dv[ip];
00146
```

5.4 dist.c 31

```
avtd2 += gsl_pow_2(dv[ip]);
00148
00149
               /* Calculate relative transport deviations... */
00150
               if (f > 2) {
00151
00152
                  /* Get trajectory lengths... */
                  geo2cart(0, lon1[ip], lat1[ip], x0);
00153
00154
                  lh1[ip] += DIST(x0, x1);
00155
                  lv1[ip] += fabs(Z(p1[ip]) - Z(atm1->p[ip]));
00156
                  geo2cart(0, lon2[ip], lat2[ip], x0);
00157
                  lh2[ip] += DIST(x0, x2);
lv2[ip] += fabs(Z(p2[ip]) - Z(atm2->p[ip]));
00158
00159
00160
00161
                  /\star Get relative transport devations... \star/
                  if (lh1[ip] + lh2[ip] > 0) {
  aux = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00162
00163
                    rhtd += aux;
00164
                    rhtd2 += gsl_pow_2(aux);
00165
00166
00167
                  if (lv1[ip] + 1v2[ip] > 0) {
00168
                      200. * fabs(Z(atm1->p[ip]) - Z(atm2->p[ip])) / (lv1[ip] +
00169
00170
                                                                                     lv2[ip]);
00171
                    rvtd += aux;
00172
                    rvtd2 += gsl_pow_2(aux);
00173
00174
00175
00176
               /\star Save positions of air parcels... \star/
               lon1[ip] = atml->lon[ip];
lat1[ip] = atml->lat[ip];
00177
00178
00179
               p1[ip] = atm1->p[ip];
00180
               lon2[ip] = atm2->lon[ip];
lat2[ip] = atm2->lat[ip];
00181
00182
              p2[ip] = atm2->p[ip];
00183
00184
00185
00186
             /\star Get indices of trajectories with maximum errors... \star/
00187
            iph = (int) gsl_stats_max_index(dh, 1, (size_t) atm1->np);
00188
            ipv = (int) gsl_stats_max_index(dv, 1, (size_t) atm1->np);
00189
00190
             /* Sort distances to calculate percentiles... */
            gsl_sort(dh, 1, (size_t) atm1->np);
gsl_sort(dv, 1, (size_t) atm1->np);
00191
00192
00193
            /* Get date from filename... */
for (i = (int) strlen(argv[f]) - 1; argv[f][i] != '/' || i == 0; i--);
name = strtok(&(argv[f][i]), "_");
00194
00195
00196
            year = strtok(NULL, "_");
mon = strtok(NULL, "_");
day = strtok(NULL, "_");
00197
00198
00199
            hour = strtok(NULL, "_");
name = strtok(NULL, "_");
min = strtok(name, ".");
00200
00201
                                                /* TODO: Why another "name" here? */
00202
00203
            time2jsec(atoi(year), atoi(mon), atoi(day), atoi(hour), atoi(min), 0, 0,
00204
                         &t);
00205
            00206
00207
                       " %g %g %g %g %g %g %g %g %d %g %g\n", t,
00208
00209
                       ahtd / atml->np,
00210
                       sqrt(ahtd2 / atm1->np - gsl_pow_2(ahtd / atm1->np)),
00211
                       dh[0], dh[atm1->np / 10], dh[atm1->np / 4], dh[atm1->np / 2],
                      dh[atml->np - atml->np / 4], dh[atml->np - atml->np / 10],
dh[atml->np - 1], iph, rhtd / atml->np,
sqrt(rhtd2 / atml->np - gsl_pow_2(rhtd / atml->np)),
00212
00213
00214
00215
                       avtd / atm1->np,
                       sqrt(avtd2 / atm1->np - gsl_pow_2(avtd / atm1->np)),
                      dv[0], dv[atml->np / 10], dv[atml->np / 4], dv[atml->np / 2],
dv[atml->np - atml->np / 4], dv[atml->np / 10],
dv[atml->np - atml->np / 4], dv[atml->np - atml->np / 10],
dv[atml->np - 1], ipv, rvtd / atml->np,
sqrt(rvtd2 / atml->np - gsl_pow_2(rvtd / atml->np)));
00217
00218
00219
00220
00221
00222
00223
          /* Close file... */
00224
         fclose(out);
00225
00226
         /* Free... */
00227
         free(atm1);
00228
          free(atm2);
00229
          free (lon1);
00230
          free(lat1);
00231
         free(p1);
00232
          free (lh1);
00233
         free(lv1);
```

```
00234
       free(lon2);
00235
       free(lat2);
00236
       free(p2);
00237
       free(lh2);
00238
       free(lv2);
00239
       free (dh);
       free(dv);
00241
00242
       return EXIT_SUCCESS;
00243 }
```

#### 5.5 extract.c File Reference

Extract single trajectory from atmospheric data files.

#### **Functions**

• int main (int argc, char \*argv[])

## 5.5.1 Detailed Description

Extract single trajectory from atmospheric data files.

Definition in file extract.c.

#### 5.5.2 Function Documentation

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

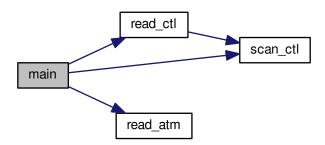
Definition at line 28 of file extract.c.

```
00030
00031
00032
         ctl_t ctl;
00033
00034
        atm_t *atm;
00035
00036
        FILE *in, *out;
00037
00038
        int f, ip, iq;
00039
00040
         /* Allocate... */
00041
         ALLOC(atm, atm_t, 1);
00042
00043
         /* Check arguments... */
00044
         if (argc < 4)
00045
           ERRMSG("Give parameters: <ctl> <outfile> <atml> [<atm2> ...]");
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
00051
         /* Write info... */
00052
        printf("Write trajectory data: sn', argv[2]);
00053
00054
         /* Create output file... */
00055
        if (!(out = fopen(argv[2], "w")))
00056
           ERRMSG("Cannot create file!");
00057
00058
         /* Write header... */
00059 fprintf(out,
                  "# $1 = time [s]\n"
"# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00060
00061
00062
00063
        for (iq = 0; iq < ctl.nq; iq++)</pre>
```

5.6 extract.c 33

```
fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
        ctl.qnt_unit[iq]);
fprintf(out, "\n");
00065
00066
00067
        /* Loop over files... */
for (f = 3; f < argc; f++) {</pre>
00068
00069
00070
00071
          /* Read atmopheric data... */
          if (!(in = fopen(argv[f], "r")))
00072
00073
            continue;
00074
          else
00075
           fclose(in);
00076
          read_atm(argv[f], &ctl, atm);
00077
          00078
00079
                  Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
08000
          for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");</pre>
00081
00082
00083
             fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00084
00085
          fprintf(out, "\n");
00086
00087
00088
        /* Close file... */
00089
        fclose(out);
00090
        /* Free... */
00091
00092
        free(atm);
00093
00094
        return EXIT SUCCESS:
00095 }
```

Here is the call graph for this function:



## 5.6 extract.c

```
00001 /*
00002
         This file is part of MPTRAC.
00003
00004
         \ensuremath{\mathsf{MPTRAC}} is free software: you can redistribute it and/or modify
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
         \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/>.
00016
00017
         Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
```

```
00025 #include "libtrac.h"
00026 #include <gsl/gsl_sort.h>
00027
00028 int main(
00029
        int argc,
00030
        char *argv[]) {
00031
00032
        ctl_t ctl;
00033
00034
        atm t *atm;
00035
        FILE *in, *out;
00036
00037
00038
        int f, ip, iq;
00039
00040
         /* Allocate... */
00041
        ALLOC(atm, atm_t, 1);
00042
00043
        /* Check arguments... */
00044
        if (argc < 4)
00045
           ERRMSG("Give parameters: <ctl> <outfile> <atm1> [<atm2> ...]");
00046
00047
        /* Read control parameters... */
        read_ctl(argv[1], argc, argv, &ctl);
ip = (int) scan_ctl(argv[1], argc, argv, "EXTRACT_IP", -1, "0", NULL);
00048
00049
00050
00051
         /\star Write info... \star/
00052
        printf("Write trajectory data: s\n", argv[2]);
00053
         /* Create output file... */
00054
        if (!(out = fopen(argv[2], "w")))
00055
00056
           ERRMSG("Cannot create file!");
00057
00058
        /* Write header... */
00059
        fprintf(out,
00060
                  "# $1 = time [s]\n"
                  "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00061
00062
        for (iq = 0; iq < ctl.nq; iq++)
    fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],</pre>
00063
00064
        ctl.qnt_unit[iq]);
fprintf(out, "\n");
00065
00066
00067
00068
         /* Loop over files... */
00069
         for (f = 3; f < argc; f++) {</pre>
00070
           /* Read atmopheric data... */
if (!(in = fopen(argv[f], "r")))
00071
00072
00073
             continue;
00074
           else
00075
             fclose(in);
00076
           read_atm(argv[f], &ctl, atm);
00077
           /* Write data... */
fprintf(out, "%.2f %g %g %g", atm->time[ip],
00078
00079
08000
                   Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
           for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");</pre>
00081
00082
00083
             fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00084
00085
          fprintf(out, "\n");
00086
00087
00088
         /* Close file... */
00089
         fclose(out);
00090
00091
         /* Free... */
00092
        free(atm);
00093
00094
         return EXIT_SUCCESS;
00095 }
```

# 5.7 init.c File Reference

Create atmospheric data file with initial air parcel positions.

# **Functions**

• int main (int argc, char \*argv[])

5.7 init.c File Reference 35

### 5.7.1 Detailed Description

Create atmospheric data file with initial air parcel positions.

Definition in file init.c.

### 5.7.2 Function Documentation

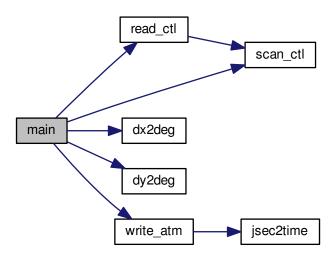
### 5.7.2.1 int main ( int *argc*, char \* *argv[*])

Definition at line 27 of file init.c.

```
00029
00031
               atm_t *atm;
00032
00033
              ctl_t ctl;
00034
00035
              asl rna *rna:
00036
00037
               double dt, dz, dlon, dlat, lat0, lat1, lon0, lon1, t0, t1, z0, z1,
                   t, z, lon, lat, st, sz, slon, slat, sx, ut, uz, ulon, ulat, m;
00038
00039
00040
              int ip, irep, rep;
00041
00042
                  * Allocate... */
00043
               ALLOC(atm, atm_t, 1);
00044
00045
                /\star Check arguments... \star/
00046
               if (argc < 3)
                   ERRMSG("Give parameters: <ctl> <atm_out>");
00047
00048
              /* Read control parameters... */
              /* 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_TI", -1, "0", NULL);
dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
z0 = scan_ctl(argv[1], argc, argv, "INIT_2O", -1, "0", NULL);
00050
00051
00052
00053
00054
               z1 = scan_ctl(argv[1], argc, argv, "INIT_21", -1, "0", NULL);
dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);
              dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);
lon0 = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "0", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "INIT_LON0", -1, "0", NULL);
dlon = scan_ctl(argv[1], argc, argv, "INIT_LOND", -1, "1", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "INIT_LAT0", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT1", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT1", -1, "0", NULL);
st = scan_ctl(argv[1], argc, argv, "INIT_ST", -1, "0", NULL);
sz = scan_ctl(argv[1], argc, argv, "INIT_SZ", -1, "0", NULL);
slon = scan_ctl(argv[1], argc, argv, "INIT_SZ", -1, "0", NULL);
slat = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
sx = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_UT", -1, "0", NULL);
ulon = scan_ctl(argv[1], argc, argv, "INIT_UT", -1, "0", NULL);
ulon = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
rep = (int) scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
m = scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
00056
00057
00058
00059
00060
00061
00062
00063
00064
00065
00066
00067
00068
00069
00070
00071
00072
00073
00075
                /* Initialize random number generator... */
00076
               gsl_rng_env_setup();
00077
              rng = gsl_rng_alloc(gsl_rng_default);
00078
00079
                /* Create grid... */
               for (t = t0; t <= t1; t += dt)
08000
                  for (z = z0; z \le z1; z += dz)
00081
00082
                      for (lon = lon0; lon <= lon1; lon += dlon)</pre>
                           for (lat = lat0; lat <= lat1; lat += dlat)</pre>
00083
00084
                               for (irep = 0; irep < rep; irep++) {</pre>
00085
00086
                                   /* Set position... */
                                   atm->time[atm->np]
00087
00088
                                      = (t + gsl_ran_gaussian_ziggurat(rng, st / 2.3548)
00089
                                            + ut * (gsl_rng_uniform(rng) - 0.5));
00090
                                   atm->p[atm->np]
00091
                                      = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
00092
                                                + uz * (gsl_rng_uniform(rng) - 0.5));
00093
                                   atm->lon[atm->np]
```

```
= (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));
00095
00096
00097
                     atm->lat[atm->np]
                        = (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));
00098
00099
00100
00101
00102
                      /* Set particle counter... */
                      if ((++atm->np) >= NP)
00103
                        ERRMSG("Too many particles!");
00104
00105
00106
00107
          /* Check number of air parcels... */
00108
         if (atm->np <= 0)
           ERRMSG("Did not create any air parcels!");
00109
00110
00111
          /* Initialize mass... */
         if (ctl.qnt_m >= 0)
00112
          for (ip = 0; ip < atm->np; ip++)
00113
00114
              atm->q[ctl.qnt_m][ip] = m / atm->np;
00115
         /* Save data... */
write_atm(argv[2], &ctl, atm, t0);
00116
00117
00118
00119
         /* Free... */
00120
         gsl_rng_free(rng);
00121
         free(atm);
00122
00123
         return EXIT_SUCCESS;
00124 }
```

Here is the call graph for this function:



# 5.8 init.c

```
00001 /*
00002 This file is part of MPTRAC.
00003
00004 MPTRAC is free software: you can redistribute it and/or modify
00005 it under the terms of the GNU General Public License as published by
00006 the Free Software Foundation, either version 3 of the License, or
00007 (at your option) any later version.
00008
00009 MPTRAC is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
```

5.8 init.c 37

```
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
             Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028
             int argc,
00029
             char *argv[]) {
00030
00031
             atm_t *atm;
00032
00033
             ctl t ctl;
00034
00035
             gsl_rng *rng;
00036
00037
             double dt, dz, dlon, dlat, lat0, lat1, lon0, lon1, t0, t1, z0, z1,
00038
                t, z, lon, lat, st, sz, slon, slat, sx, ut, uz, ulon, ulat, m;
00039
00040
             int ip, irep, rep;
00041
              /* Allocate... */
00042
00043
             ALLOC(atm, atm_t, 1);
00044
00045
              /* Check arguments... */
00046
             if (argc < 3)
00047
                ERRMSG("Give parameters: <ctl> <atm_out>");
00048
             /\star Read control parameters... \star/
00049
             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
            dt = scan_ctl(argv[1], argc, argv, "INIT_DIT", -1, "1", NULL);
z0 = scan_ctl(argv[1], argc, argv, "INIT_Z0", -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_LON0", -1, "0", NULL);
dlon1 = scan_ctl(argv[1], argc, argv, "INIT_LON1", -1, "0", NULL);
dlon2 = scan_ctl(argv[1], argc, argv, "INIT_LON1", -1, "0", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "INIT_LAT0", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT1", -1, "0", NULL);
dlat = scan_ctl(argv[1], argc, argv, "INIT_DLAT", -1, "1", NULL);
st = scan_ctl(argv[1], argc, argv, "INIT_ST", -1, "0", NULL);
sz = scan_ctl(argv[1], argc, argv, "INIT_SZ", -1, "0", NULL);
slon = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
sx = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -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);
ulon = scan_ctl(argv[1], argc, argv, "INIT_UT", -1, "0", NULL);
ulat = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
ulat = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
m = scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
00054
             z0 = scan_ctl(argv[1], argc, argv, "INIT_ZO", -1, "0", NULL);
00055
00056
00057
00058
00059
00060
00061
00062
00063
00064
00065
00066
00067
00068
00069
00070
00071
00073
00074
00075
             /* Initialize random number generator... */
00076
             gsl_rng_env_setup();
00077
             rng = gsl_rng_alloc(gsl_rng_default);
00078
00079
              /* Create grid... */
00080
             for (t = t0; t <= t1; t += dt)
00081
                for (z = z0; z \le z1; z += dz)
00082
                     for (lon = lon0; lon <= lon1; lon += dlon)
                        for (lat = lat0; lat <= lat1; lat += dlat)</pre>
00083
00084
                           for (irep = 0; irep < rep; irep++) {</pre>
00086
                                /* Set position... */
00087
                               atm->time[atm->np]
00088
                                   = (t + gsl_ran_gaussian_ziggurat(rng, st / 2.3548)
00089
                                        + ut * (gsl_rng_uniform(rng) - 0.5));
00090
                               atm->p[atm->np]
00091
                                  = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
00092
                                          + uz * (gsl_rng_uniform(rng) - 0.5));
00093
                               atm->lon[atm->np]
                                  = (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));
00094
00095
00096
00097
                               atm->lat[atm->np]
00098
                                   = (lat + gsl_ran_gaussian_ziggurat(rng, slat / 2.3548)
00099
                                         + gsl_ran_gaussian_ziggurat(rng, dy2deg(sx) / 2.3548)
00100
                                         + ulat * (gsl_rng_uniform(rng) - 0.5));
00101
00102
                               /* Set particle counter... */
```

```
if ((++atm->np) >= NP)
00104
                       ERRMSG("Too many particles!");
00105
00106
         /* Check number of air parcels... */
if (atm->np <= 0)</pre>
00107
00108
00109
           ERRMSG("Did not create any air parcels!");
00110
00111
         /* Initialize mass... */
         if (ctl.qnt_m >= 0)
  for (ip = 0; ip < atm->np; ip++)
    atm->q[ctl.qnt_m][ip] = m / atm->np;
00112
00113
00114
00115
00116
00117
         write_atm(argv[2], &ctl, atm, t0);
00118
00119
         /* Free... */
         gsl_rng_free(rng);
free(atm);
00120
00122
00123
         return EXIT_SUCCESS;
00124 }
```

# 5.9 jsec2time.c File Reference

Convert Julian seconds to date.

### **Functions**

• int main (int argc, char \*argv[])

# 5.9.1 Detailed Description

Convert Julian seconds to date.

Definition in file jsec2time.c.

## 5.9.2 Function Documentation

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

Definition at line 27 of file jsec2time.c.

```
00029
00030
00031
         double jsec, remain;
00032
00033
         int day, hour, min, mon, sec, year;
00034
00035
         /* Check arguments... */
00036
           ERRMSG("Give parameters: <jsec>");
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;
00047 }
```

5.10 jsec2time.c 39

Here is the call graph for this function:



# 5.10 jsec2time.c

```
00001 /*
        This file is part of MPTRAC.
00002
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
00006
        the Free Software Foundation, either version 3 of the License, or
00007
        (at your option) any later version.
80000
00009
        MPTRAC is distributed in the hope that it will be useful,
        but WITHOUT ANY WARRANTY; without even the implied warranty of
00010
00011
        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
        GNU General Public License for more details.
00013
        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
        Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
        int argc,
00029
        char *argv[]) {
00030
00031
        double jsec, remain;
00032
00033
        int day, hour, min, mon, sec, year;
00034
00035
        /* Check arguments... */
00036
00037
          ERRMSG("Give parameters: <jsec>");
00038
00039
        /* Read arguments... */
00040
        jsec = atof(argv[1]);
00041
00042
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 }
```

# 5.11 libtrac.c File Reference

MPTRAC library definitions.

# **Functions**

- void cart2geo (double \*x, double \*z, double \*lon, double \*lat)

  Convert Cartesian coordinates to geolocation.
- 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 \*t, double \*u, double \*v, double \*w, 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 \*t, double \*u, double \*v, double \*w, 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\_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.

 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.

- double tropopause (double t, double lat)
- 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.

### 5.11.1 Detailed Description

MPTRAC library definitions.

Definition in file libtrac.c.

### 5.11.2 Function Documentation

```
5.11.2.1 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 }
```

5.11.2.2 double deg2dx ( double dlon, double lat )

Convert degrees to horizontal distance.

Definition at line 45 of file libtrac.c.

```
00047

00048

00049 return dlon * M_PI * RE / 180. * cos(lat / 180. * M_PI);

00050 }
```

```
5.11.2.3 double deg2dy ( double dlat )
```

Convert degrees to horizontal distance.

Definition at line 54 of file libtrac.c.

```
00055 {
00056
00057 return dlat * M_PI * RE / 180.;
00058 }
```

5.11.2.4 double dp2dz ( double dp, double p )

Convert pressure to vertical distance.

Definition at line 62 of file libtrac.c.

```
00064 {
00065
00066 return -dp * H0 / p;
00067 }
```

5.11.2.5 double dx2deg ( double dx, double lat )

Convert horizontal distance to degrees.

Definition at line 71 of file libtrac.c.

5.11.2.6 double dy2deg ( double dy )

Convert horizontal distance to degrees.

Definition at line 84 of file libtrac.c.

5.11.2.7 double dz2dp ( double dz, double p )

Convert vertical distance to pressure.

Definition at line 92 of file libtrac.c.

```
00094 {
00095
00096 return -dz * p / H0;
00097 }
```

```
5.11.2.8 void geo2cart (double z, double lon, double lat, double *x)
```

Convert geolocation to Cartesian coordinates.

Definition at line 101 of file libtrac.c.

```
00105 {
00106
00107 double radius;
00108
00109 radius = z + RE;
00110 x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
00111 x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
00112 x[2] = radius * sin(lat / 180 * M_PI);
```

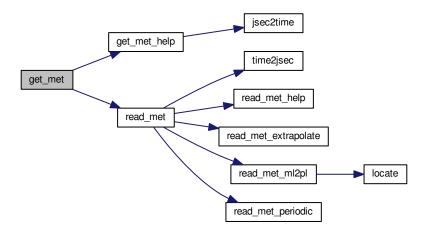
```
5.11.2.9 void get_met ( ctl_t * ctl, char * metbase, double t, met_t * met0, met_t * met1 )
```

Get meteorological data for given timestep.

Definition at line 117 of file libtrac.c.

```
00122
00123
00124
        char filename[LEN];
00125
00126
        static int init;
00127
00128
        /* Init... */
00129
        if (!init) {
00130
          init = 1;
00131
          get_met_help(t, -1, metbase, ctl->dt_met, filename);
00132
          read_met(ctl, filename, met0);
00133
00134
00135
          get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
     dt_met, filename);
00136
         read_met(ctl, filename, met1);
00137
00138
00139
        /* Read new data for forward trajectories... */
00140
        if (t > met1->time && ctl->direction == 1) {
00141
        memcpy(met0, met1, sizeof(met_t));
          get_met_help(t, 1, metbase, ctl->dt_met, filename);
read_met(ctl, filename, met1);
00142
00143
00144
00145
00146
        /* Read new data for backward trajectories... */
00147
        if (t < met0->time && ctl->direction == -1) {
00148
        memcpy(met1, met0, sizeof(met_t));
          get_met_help(t, -1, metbase, ctl->dt_met, filename);
read_met(ctl, filename, met0);
00149
00150
00151
00152 }
```

Here is the call graph for this function:



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

Get meteorological data for timestep.

Definition at line 156 of file libtrac.c.

```
00161
00162
00163
      double t6, r;
00164
00165
      int year, mon, day, hour, min, sec;
00166
      /* Round time to fixed intervals... */
if (direct == -1)
00167
00168
00169
        t6 = floor(t / dt_met) * dt_met;
00170
00171
        t6 = ceil(t / dt_met) * dt_met;
00172
00173
      /* Decode time... */
00174
      jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00175
      00176
00177
00178 }
```

Here is the call graph for this function:



5.11.2.11 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 182 of file libtrac.c.

```
{
00189
00190
         double aux00, aux01, aux10, aux11;
00191
00192
         /* Set variables...
        aux00 = array[ix][iy];
00193
        aux01 = array[ix][iy + 1];
aux10 = array[ix + 1][iy];
00194
00195
00196
         aux11 = array[ix + 1][iy + 1];
00197
00198
         /* Interpolate horizontally... */
        aux00 = wy * (aux00 - aux01) + aux01;

aux11 = wy * (aux10 - aux11) + aux11;
00199
00200
00201
         *var = wx * (aux00 - aux11) + aux11;
00202 }
```

5.11.2.12 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 206 of file libtrac.c.

```
00214
                         {
00216
         double aux00, aux01, aux10, aux11;
00217
00218
         /* Interpolate vertically... */
         aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00219
         + array[ix][iy][ip + 1];
aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
+ array[ix][iy + 1][ip + 1];
00220
00221
00222
00223
         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])
00224
00225
00226
           + array[ix + 1][iy + 1][ip + 1];
00227
         /* Interpolate horizontally... */
        aux00 = wy * (aux00 - aux01) + aux01;
aux11 = wy * (aux10 - aux11) + aux11;
00229
00230
00231
         *var = wx * (aux00 - aux11) + aux11;
00232 }
```

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

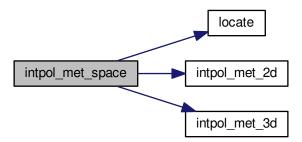
Spatial interpolation of meteorological data.

Definition at line 236 of file libtrac.c.

```
00248
00249
        double wp, wx, wy;
00250
00251
        int ip, ix, iy;
00252
        /* Check longitude... */
if (met->lon[met->nx - 1] > 180 && lon < 0)
00253
00254
00255
          lon += 360;
00256
00257
        /* Get indices... */
00258
       ip = locate(met->p, met->np, p);
00259
       ix = locate(met->lon, met->nx, lon);
        iy = locate(met->lat, met->ny, lat);
```

```
00261
        /* Get weights... */
00262
        wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
00263
        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]);
00264
00265
00266
00267
        /* Interpolate... */
00268
        if (ps != NULL)
00269
          intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
00270
        if (t != NULL)
00271
          intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
00272
        if (u != NULL)
00273
          intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
00274
        if (v != NULL)
00275
          intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
00276
        if (w != NULL)
00277
          intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
00278
        if (h2o != NULL)
          intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
00280
        if (o3 != NULL)
00281
          intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00282 }
```

Here is the call graph for this function:



5.11.2.14 void intpol\_met\_time ( met\_t \* met0, met\_t \* met1, double ts, double p, double lon, double lat, double \* ps, double \* t, double \* u, double \* v, double \* w, double \* b2o, double \* o3 )

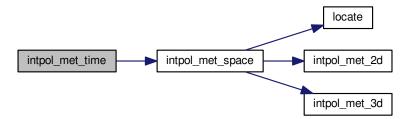
Temporal interpolation of meteorological data.

Definition at line 286 of file libtrac.c.

```
00300
00301
      double h200, h201, o30, o31, ps0, ps1, t0, t1, u0, u1, v0, v1, w0, w1, wt;
00302
00303
       /* Spatial interpolation... */
      00304
00305
00306
                      t == NULL ? NULL : &t0,
00307
                      u == NULL ? NULL : &u0,
00308
                      v == NULL ? NULL : &v0,
00309
                      w == NULL ? NULL : &w0,
                     h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00310
      00311
00312
00313
                      t == NULL ? NULL : &t1,
00314
                      u == NULL ? NULL : &u1,
                      v == NULL ? NULL : &v1,
00315
00316
                      w == NULL ? NULL : &w1,
00317
                     h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00318
```

```
/* Get weighting factor... */
00320
        wt = (met1->time - ts) / (met1->time - met0->time);
00321
00322
        /\star Interpolate... \star/
00323
        if (ps != NULL)
        *ps = wt * (ps0 - ps1) + ps1;
if (t != NULL)
00324
00326
          *t = wt * (t0 - t1) + t1;
00327
        if (u != NULL)
        *u = wt * (u0 - u1) + u1;
if (v != NULL)
00328
00329
          *v = wt * (v0 - v1) + v1;
00330
        if (w != NULL)
00331
00332
          *w = wt * (w0 - w1) + w1;
        if (h2o != NULL)
00333
        *h2o = wt * (h2o0 - h2o1) + h2o1;
if (o3 != NULL)
00334
00335
          *o3 = wt * (o30 - o31) + o31;
00336
00337 }
```

Here is the call graph for this function:



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

Convert seconds to date.

Definition at line 341 of file libtrac.c.

```
00349
                        {
00350
00351
       struct tm t0, *t1;
00352
00353
       time_t jsec0;
00354
00355
       t0.tm_year = 100;
        t0.tm_mon = 0;
00356
00357
        t0.tm_mday = 1;
00358
        t0.tm\_hour = 0;
00359
       t0.tm_min = 0;
00360
       t0.tm\_sec = 0;
00361
        jsec0 = (time_t) jsec + timegm(&t0);
00362
00363
       t1 = gmtime(&jsec0);
00364
00365
       *year = t1->tm_year + 1900;
00366
       *mon = t1->tm_mon + 1;
00367
        *day = t1->tm_mday;
00368
        *hour = t1->tm_hour;
       *min = t1->tm_min;
00369
00370
       *sec = t1->tm_sec;
00371
       *remain = jsec - floor(jsec);
00372 }
```

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

Find array index.

Definition at line 376 of file libtrac.c.

```
00379
                   {
00380
00381
        int i, ilo, ihi;
00382
00383
        ilo = 0;
        ihi = n - 1;
i = (ihi + ilo) >> 1;
00384
00385
00386
        if (xx[i] < xx[i + 1])
00387
         while (ihi > ilo + 1) {
  i = (ihi + ilo) >> 1;
00388
00389
00390
             if (xx[i] > x)
00391
              ihi = i;
00392
             else
00393
              ilo = i;
00394
        } else
00395
          while (ihi > ilo + 1) {
00396
           i = (ihi + ilo) >> 1;
            if (xx[i] \le x)
00397
00398
              ihi = i;
            else
00399
00400
               ilo = i;
00401
         }
00402
00403
        return ilo;
00404 }
```

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

Read atmospheric data.

Definition at line 408 of file libtrac.c.

```
00411
00412
00413
          FILE *in;
00414
00415
          char line[LEN], *tok;
00416
00417
          int iq;
00418
          /* Init... */
atm->np = 0;
00419
00420
00421
00422
           /* Write info... */
00423
          printf("Read atmospheric data: sn', filename);
00424
00425
          /* Open file... */
          if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");
00426
00428
00429
           /* Read line... */
           while (fgets(line, LEN, in)) {
00430
00431
             /* 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]);
00432
00433
00434
00435
00436
00437
00438
00439
00440
              /* Convert altitude to pressure... */
00441
             atm->p[atm->np] = P(atm->p[atm->np]);
00442
             /* Increment data point counter... */
if ((++atm->np) > NP)
00443
00444
                ERRMSG("Too many data points!");
00445
00446
00447
00448
           /* Close file... */
00449
           fclose(in);
00450
00451
           /\star Check number of points... \star/
00452
           if (atm->np < 1)
00453
             ERRMSG("Can not read any data!");
00454 }
```

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

Read control parameters.

Definition at line 458 of file libtrac.c.

```
00462
00463
00464
       int ip, iq;
00465
00466
       /* Write info... */
       printf("\nMassive-Parallel\ Trajectory\ Calculations\ (MPTRAC)\n"
00467
               "(executable: %s | compiled: %s, %s)\n",
00468
               argv[0], __DATE__, __TIME__);
00470
00471
       /* Initialize quantity indices... */
       ctl->qnt_ens = -1;
ctl->qnt_m = -1;
00472
00473
00474
       ctl->qnt_r = -1;
00475
       ctl->qnt_rho = -1;
       ctl->qnt_ps = -1;
00476
00477
       ctl->qnt_p = -1;
       ctl->qnt_t = -1;
00478
       ctl->qnt_u = -1;
00479
00480
       ctl->qnt_v = -1;
00481
       ctl->qnt_w = -1;
00482
       ct1->qnt_h2o = -1;
00483
       ctl->qnt_o3 = -1;
00484
       ctl->qnt\_theta = -1;
       ctl->qnt_pv = -1;
00485
00486
       ctl->qnt\_tice = -1;
00487
       ctl->qnt\_tsts = -1;
       ctl->qnt_tnat = -1;
00489
       ctl \rightarrow qnt_stat = -1;
00490
00491
        /* Read quantities... */
       ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
00492
00493
       for (iq = 0; iq < ctl->nq; iq++) {
00494
00495
          /* Read quantity name and format...
         00496
00497
00498
00499
00500
          /\star Try to identify quantity... \star/
00501
          if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
00502
           ctl->qnt_ens = iq;
00503
           sprintf(ctl->qnt_unit[iq], "-");
00504
         } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
           ctl->qnt_m = iq;
00505
           sprintf(ctl->qnt_unit[iq], "kg");
00506
         } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
          ctl->qnt_r = iq;
00508
00509
           sprintf(ctl->qnt_unit[iq], "m");
         } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
ctl->qnt_rho = iq;
00510
00511
           sprintf(ctl->qnt_unit[iq], "kg/m^3");
00513
         } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
00514
           ctl->qnt_ps = iq;
00515
           sprintf(ctl->qnt_unit[iq], "hPa");
         } else if (strcmp(ct1->qnt_name[iq], "p") == 0) {
ct1->qnt_p = iq;
00516
00517
           sprintf(ctl->qnt_unit[iq], "hPa");
00518
         } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
00520
          ctl->qnt_t = iq;
00521
           sprintf(ctl->qnt_unit[iq], "K");
00522
         } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
           ctl->qnt_u = iq;
sprintf(ctl->qnt_unit[iq], "m/s");
00523
00524
00525
         } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
           ctl->qnt_v = iq;
00527
           sprintf(ctl->qnt_unit[iq], "m/s");
          } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
00528
           ctl->qnt_w = iq;
00529
           sprintf(ctl->qnt_unit[iq], "hPa/s");
00530
         } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
          ctl->qnt_h2o = iq;
00532
00533
           sprintf(ctl->qnt_unit[iq], "1");
00534
         } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
00535
           ctl->qnt_o3 = iq;
           sprintf(ctl->qnt_unit[iq], "1");
00536
00537
         } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
           ctl->qnt_theta = iq;
```

```
sprintf(ctl->qnt_unit[iq], "K");
00540
           } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
00541
             ctl->qnt_pv = iq;
00542
             sprintf(ctl->qnt_unit[iq], "PVU");
           } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
00543
             ctl->qnt_tice = iq;
00544
             sprintf(ctl->qnt_unit[iq], "K");
           } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
00546
             ctl->qnt_tsts = iq;
00547
00548
              sprintf(ctl->qnt_unit[iq], "K");
           } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
00549
00550
             ctl->qnt_tnat = iq;
             sprintf(ctl->qnt_unit[iq], "K");
00551
00552
           } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
00553
             ctl->qnt_stat = iq;
00554
              sprintf(ctl->qnt_unit[iq], "-");
00555
           } else
00556
              scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00558
00559
          /* Time steps of simulation... */
         ctl->direction =
00560
         (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
if (ctl->direction != -1 && ctl->direction != 1)
   ERRMSG("Set DIRECTION to -1 or 1!");
00561
00562
00563
00564
         ctl->t_start =
00565
           scan_ctl(filename, argc, argv, "T_START", -1, "-1e100", NULL);
         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);
00566
00567
00568
00569
         /* Meteorological data... */
         ctl->d_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
00570
00571
00572
         if (ctl->met_np > EP)
00573
           ERRMSG("Too many levels!");
         for (ip = 0; ip < ctl->met_np; ip++)
  ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
00574
00575
00576
00577
          /* Isosurface parameters... */
00578
         ctl->isosurf
         = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00579
00580
00581
00582
         /* Diffusion parameters... */
00583
         ctl->turb_dx_trop
00584
           = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00585
         ctl->turb_dx_strat
           = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00586
00587
         ctl->turb dz trop
00588
           = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
         ctl->turb_dz_strat
00590
            = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
         ctl->turb_meso
00591
00592
           scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00593
00594
         /* Life time of particles... */
        ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
00595
00596
        ctl->tdec_strat =
00597
           scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00598
00599
         /* PSC analysis... */
         ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
00600
         ctl->psc_hno3 =
00601
00602
           scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
00603
         /* Output of atmospheric data... */
scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
00604
00605
       atm basename);
00606
        scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00607
         ctl->atm_dt_out
00608
           scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00609
         ctl->atm filter
00610
           (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
00611
00612
         /* Output of CSI data... */
         scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
00614
        ctl->csi_dt_out =
        scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
scan_ctl(filename, argc, argv, "CSI_DBSFILE", -1, "obs.tab",
00615
00616
00617
                   ctl->csi obsfile);
00618
        ctl->csi_obsmin =
           scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00619
00620
         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);
00621
00622
00623
```

```
ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
00625
           scan_ctl(filename, argc, argv, "CSI_LONO", -1, "-180", NULL);
00626
         ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
00627
00628
         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);
00629
00630
00631
         ctl->csi_ny =
00632
00633
            (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
00634
        /* Output of ensemble data... */
scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
00635
00636
      ens_basename);
00637
         /\!\star Output of grid data... \star/
00638
         scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
00639
00640
                    ctl->grid basename);
         scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
00641
      grid_gpfile);
00642 ctl->grid_dt_out =
00643
           scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
00644
         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);
00645
00646
00648
         ctl->grid_nz =
00649
            (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00650
         ctl->grid lon0 :
           scan_ctl(filename, argc, argv, "GRID_LONO", -1, "-180", NULL);
00651
00652
         ctl->grid lon1 =
00653
           scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
00654
         ctl->grid_nx =
00655
           (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00656
         ctl->grid lat0 =
           scan_ctl(filename, argc, argv, "GRID_LATO", -1, "-90", NULL);
00657
00658
         ctl->grid lat1 =
00659
           scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
00660
         ctl->grid_ny =
00661
           (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
00662
00663
         /* Output of profile data... */
         00664
00665
prof_obsfile);

00667 ctl->prof_c^
00666
         scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
         ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
00668
00669
         ctl->prof nz =
00670
           (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
00671
         ctl->prof_lon0 =
00672
            scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
         ctl->prof_lon1
00673
00674
           scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
00675
         ctl->prof nx =
00676
            (int) scan ctl(filename, argc, argv, "PROF NX", -1, "360", NULL);
00677
         ctl->prof_lat0 =
00678
            scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
00679
         ctl->prof_lat1 =
00680
           scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
00681
         ctl->prof ny =
00682
           (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00683
00684
         /* Output of station data... */
         scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
00685
00686
                   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);
00687
00688
00689
00690 }
```

Here is the call graph for this function:



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

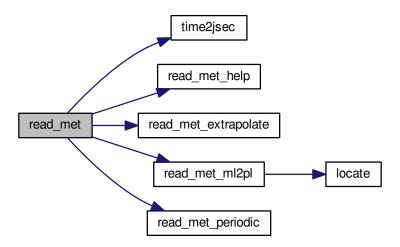
Read meteorological data file.

Definition at line 694 of file libtrac.c.

```
00697
00698
00699
         char tstr[10];
00700
00701
         static float help[EX * EY];
00702
00703
         int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00704
00705
         size_t np, nx, ny;
00706
00707
         /* Write info... */
00708
         printf("Read meteorological data: %s\n", filename);
00709
00710
         /* Get time from filename... */
00711
         sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
         year = atoi(tstr);
sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00712
00713
00714
         mon = atoi(tstr);
00715
         sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00716
         day = atoi(tstr);
00717
         sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00718
         hour = atoi(tstr);
00719
         time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00720
         /* Open netCDF file... */
00721
00722
         NC(nc_open(filename, NC_NOWRITE, &ncid));
00723
         /* Get dimensions... */
NC(nc_inq_dimid(ncid, "lon", &dimid));
NC(nc_inq_dimlen(ncid, dimid, &nx));
00724
00725
00726
00727
         if (nx > EX)
00728
           ERRMSG("Too many longitudes!");
00729
         NC(nc_inq_dimid(ncid, "lat", &dimid));
NC(nc_inq_dimlen(ncid, dimid, &ny));
if (ny > EY)
00730
00731
00732
00733
           ERRMSG("Too many latitudes!");
00734
00735
         NC(nc_inq_dimid(ncid, "lev", &dimid));
00736
         NC(nc_inq_dimlen(ncid, dimid, &np));
         if (np > EP)
00737
00738
           ERRMSG("Too many levels!");
00739
00740
         /* Store dimensions... */
         met->np = (int) np;
met->nx = (int) nx;
00741
00742
00743
         met->ny = (int) ny;
00744
00745
         /* Get horizontal grid... */
NC(nc_inq_varid(ncid, "lon", &varid));
00746
00747
         NC(nc_get_var_double(ncid, varid, met->lon));
00748
         NC(nc_inq_varid(ncid, "lat", &varid));
00749
         NC(nc_get_var_double(ncid, varid, met->lat));
00750
00751
         /* 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);
00752
00753
```

```
read_met_help(ncid, "v", "V", met, met->v, 1.0);
read_met_help(ncid, "w", "W", met, met->w, 0.01f);
read_met_help(ncid, "q", "Q", met, met->help(ncid, "o3", "03", met, met->o3, 0.602f);
00754
00755
00756
00757
00758
00759
          /* Meteo data on pressure levels... */
00760
          if (ctl->met_np <= 0) {</pre>
00761
            /* Read pressure levels from file... */
NC(nc_inq_varid(ncid, "lev", &varid));
00762
00763
00764
            NC(nc_get_var_double(ncid, varid, met->p));
00765
            for (ip = 0; ip < met->np; ip++)
  met->p[ip] /= 100.;
00766
00767
00768
             /\star Extrapolate data for lower boundary... \star/
00769
            read_met_extrapolate(met);
00770
00771
00772
          /* Meteo data on model levels... */
00773
          else {
00774
00775
             /* Read pressure data from file... */
00776
            read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00777
00778
             /* Interpolate from model levels to pressure levels... */
00779
            read_met_ml2pl(ctl, met, met->t);
00780
             read_met_ml2pl(ctl, met, met->u);
00781
             read_met_ml2pl(ctl, met, met->v);
00782
             read_met_ml2pl(ctl, met, met->w);
00783
             read_met_ml2pl(ctl, met, met->h2o);
00784
            read_met_ml2pl(ctl, met, met->o3);
00785
00786
             /* Set pressure levels... */
            met->np = ctl->met_np;
for (ip = 0; ip < met->np; ip++)
00787
00788
00789
               met->p[ip] = ctl->met_p[ip];
00790
00791
00792
          /* Check ordering of pressure levels... */
          for (ip = 1; ip < met->np; ip++)
  if (met->p[ip - 1] < met->p[ip])
00793
00794
00795
               {\tt ERRMSG("Pressure levels must be descending!");}
00796
          /* Read surface pressure... */
if (nc_ing_varid(ncid, "PS", &varid) == NC_NOERR) {
   NC(nc_get_var_float(ncid, varid, help));
00797
00798
00799
00800
             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) {
00801
00802
00803
00804
            NC(nc_get_var_float(ncid, varid, help));
            for (iy = 0; iy < met->ny; iy++)
  for (ix = 0; ix < met->nx; ix++)
00805
00806
00807
                 met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00808
          } else
00809
            for (ix = 0; ix < met->nx; ix++)
              for (iy = 0; iy < met->ny; iy++)
00811
                  met \rightarrow ps[ix][iy] = met \rightarrow p[0];
00812
00813
          /\star Create periodic boundary conditions... \star/
00814
          read_met_periodic(met);
00815
00816
           /* Close file...
          NC(nc_close(ncid));
00818 }
```

Here is the call graph for this function:



5.11.2.20 void read\_met\_extrapolate ( met\_t \* met )

Extrapolate meteorological data at lower boundary.

Definition at line 822 of file libtrac.c.

```
00823
00824
00825
         int ip, ip0, ix, iy;
00827
         /* Loop over columns... */
00828
         for (ix = 0; ix < met->nx; ix++)
           for (iy = 0; iy < met->ny; iy++) {
00829
00830
00831
               /* Find lowest valid data point... */
00832
              for (ip0 = met->np - 1; ip0 >= 0; ip0--)
00833
                 if (!gsl_finite(met->t[ix][iy][ip0])
00834
                      || !gsl_finite(met->u[ix][iy][ip0])
00835
                      || !gsl_finite(met->v[ix][iy][ip0])
00836
                      || !gsl_finite(met->w[ix][iy][ip0]))
00837
                   break;
00838
00839
               /* Extrapolate... */
              for (ip = ip0; ip >= 0; ip--) {
  met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
  met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
00840
00841
00842
                met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
00843
00844
00845
                 met \rightarrow h2o[ix][iy][ip] = met \rightarrow h2o[ix][iy][ip + 1];
00846
                 met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00847
00848
            }
00849 }
```

5.11.2.21 void read\_met\_help ( int ncid, char \* varname, char \* varname2, met\_t \* met, float dest[EX][EY][EP], float scl )

Read and convert variable from meteorological data file.

Definition at line 853 of file libtrac.c.

```
{
00860
00861
         static float help[EX * EY * EP];
00862
         int ip, ix, iy, n = 0, varid;
00863
00864
         /* Check if variable exists... */
00865
00866
         if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00867
         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00868
00869
00870
         /* Read data... */
        NC(nc_get_var_float(ncid, varid, help));
00871
00872
00873
         /* Copy and check data... */
00874
         for (ip = 0; ip < met->np; ip++)
           for (iy = 0; iy < met->ny; iy++)
  for (ix = 0; ix < met->nx; ix++) {
00875
00876
               dest[ix][iy][ip] = scl * help[n++];
                if (fabs(dest[ix][iy][ip] / scl) > 1e14)
  dest[ix][iy][ip] = GSL_NAN;
00878
00879
00880
00881 }
```

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

Convert meteorological data from model levels to pressure levels.

Definition at line 885 of file libtrac.c.

```
00888
00889
00890
       double aux[EP], p[EP], pt;
00891
00892
       int ip, ip2, ix, iy;
00893
00894
        /* Loop over columns... */
00895
        for (ix = 0; ix < met->nx; ix++)
00896
         for (iy = 0; iy < met->ny; iy++) {
00897
00898
            /* Copy pressure profile... */
for (ip = 0; ip < met->np; ip++)
00899
00900
             p[ip] = met \rightarrow pl[ix][iy][ip];
00901
            /* Interpolate... */
for (ip = 0; ip < ctl->met_np; ip++) {
   pt = ctl->met_p[ip];
00902
00903
00904
00905
              00906
               pt = p[0];
00907
              else if ((pt > p[met->np - 1] && p[1] > p[0])
                      || (pt < p[met->np - 1] && p[1] < p[0]))
00908
             00909
00910
00911
00912
00913
00914
            /* Copy data... */
for (ip = 0; ip < ctl->met_np; ip++)
00915
00916
00917
             var[ix][iy][ip] = (float) aux[ip];
00918
00919 }
```

Here is the call graph for this function:



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

Create meteorological data with periodic boundary conditions.

Definition at line 923 of file libtrac.c.

```
00924
00925
00926
         int ip, iy;
00927
00928
         /* Check longitudes... */
         if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
00930
                      + met -> lon[1] - met -> lon[0] - 360) < 0.01))
00931
00932
00933
         /* Increase longitude counter... */
         if ((++met->nx) > EX)
00934
00935
           ERRMSG("Cannot create periodic boundary conditions!");
00936
00937
         /* Set longitude... */
00938
        met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
      lon[0];
00939
00940
         /\star Loop over latitudes and pressure levels... \star/
00941
         for (iy = 0; iy < met->ny; iy++)
00942
          for (ip = 0; ip < met->np; ip++) {
00943
             met->ps[met->nx - 1][iy] = met->ps[0][iy];
             met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
00944
00945
             met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
00946
00947
00948
             met \rightarrow h2o[met \rightarrow nx - 1][iy][ip] = met \rightarrow h2o[0][iy][ip];
00949
             met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
00950
00951 }
```

5.11.2.24 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 955 of file libtrac.c.

```
00962
                        {
00963
00964
        FILE *in = NULL;
00965
00966
        char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
00967
          msg[LEN], rvarname[LEN], rval[LEN];
00968
00969
        int contain = 0, i;
00970
         /* Open file... */
00971
         if (filename[strlen(filename) - 1] != '-')
if (!(in = fopen(filename, "r")))
00972
00973
             ERRMSG("Cannot open file!");
00974
00975
00976
         /* Set full variable name... */
00977
         if (arridx >= 0) {
         sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
00978
00979
00980
        } else {
         sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
00981
00982
00983
00984
00985
         /* Read data... */
00986
         if (in != NULL)
00987
          while (fgets(line, LEN, in))
00988
             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
00989
               if (strcasecmp(rvarname, fullname1) == 0 ||
00990
                    strcasecmp(rvarname, fullname2) == 0) {
00991
                  contain = 1;
00992
                 break:
00993
00994
        for (i = 1; i < argc - 1; i++)</pre>
```

```
if (strcasecmp(argv[i], fullname1) == 0 ||
           strcasecmp(argv[i], fullname2) == 0) {
sprintf(rval, "%s", argv[i + 1]);
00996
00997
00998
           contain = 1;
00999
           break;
01000
         }
01001
01002
       /* Close file... ∗/
01003
       if (in != NULL)
01004
         fclose(in);
01005
       /* Check for missing variables... */
01006
01007
       if (!contain) {
        if (strlen(defvalue) > 0)
01008
01009
           sprintf(rval, "%s", defvalue);
01010
          sprintf(msg, "Missing variable %s!\n", fullname1);
01011
01012
           ERRMSG(msg);
01013
01014
01015
01016
       /* Write info... */
       printf("%s = %s\n", fullname1, rval);
01017
01018
01019
        /* Return values... */
01020
       if (value != NULL)
01021
         sprintf(value, "%s", rval);
01022
       return atof(rval);
01023 }
```

5.11.2.25 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 1027 of file libtrac.c.

```
01035
01036
01037
       struct tm t0, t1;
01038
01039
       t0.tm_year = 100;
01040
        t0.tm_mon = 0;
01041
       t0.tm_mday = 1;
       t0.tm_hour = 0;
01042
       t0.tm_min = 0;
01043
       t0.tm\_sec = 0;
01044
01045
01046
       t1.tm_year = year - 1900;
01047
        t1.tm_mon = mon - 1;
       t1.tm_mday = day;
01048
       t1.tm_hour = hour;
01049
01050
       t1.tm_min = min;
       t1.tm_sec = sec;
01052
01053
       *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01054 }
```

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

Measure wall-clock time.

Definition at line 1058 of file libtrac.c.

```
01061
01062
        static double starttime[NTIMER], runtime[NTIMER];
01063
01064
01065
01066
        if (id < 0 || id >= NTIMER)
01067
         ERRMSG("Too many timers!");
01068
01069
        /* Start timer... */
01070
       if (mode == 1) {
         if (starttime[id] <= 0)</pre>
```

```
starttime[id] = omp_get_wtime();
01073
01074
            ERRMSG("Timer already started!");
01075
        }
01076
01077
        /* Stop timer... */
        else if (mode == 2) {
         if (starttime[id] > 0) {
01079
01080
           runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
01081
            starttime[id] = -1;
          } else
01082
            ERRMSG("Timer not started!");
01083
01084
        }
01085
01086
        /\star Print timer...
        else if (mode == 3)
  printf("%s = %g s\n", name, runtime[id]);
01087
01088
01089 }
```

## 5.11.2.27 double tropopause ( double t, double lat )

Definition at line 1093 of file libtrac.c.

```
01096
01097
          static double doys[12]
01098
         = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01099
01100
          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,
01101
             -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
01103
01104
             -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
            15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5, 45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5, 75, 77.5, 80, 82.5, 85, 87.5, 90
01105
01106
01107
01108
01109
01110
          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,
01111
01112
01113
                    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,
01114
01115
01116
                    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, 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,
01117
01118
01119
01120
           150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
01122
           98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
01123
           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, 284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2, 287.5, 286.2, 285.8},
01124
01125
          {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
01127
           297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
01128
01129
           161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
01130
           100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
           99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8, 186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
01131
01132
           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},
01134
01135
          {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,
01136
01137
           102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
01138
           99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
01139
           148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
01140
           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},
01141
01142
          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,
01143
01144
           205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7,
01145
                                                                                   104.1,
           101.8, 101.4, 101.1, 101, 101, 101.1, 101.2, 101.5, 101.9,
01146
            102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
01147
01148
           165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
01149
           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},
01150
01151
          {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, 106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
01154
01155
                127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245, 251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9, 308.5, 312.2, 313.1, 313.3}, {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
01156
01157
01158
01159
                   187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
01160
01161
                  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, 275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8),
01162
01163
01164
01165
01166
01167
                {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,
01168
01169
01170
01171
                  120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
                  230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
01173
01174
                  278.2, 282.6, 287.4, 290.9, 292.5, 293},
01175
                {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,
01176
01177
                  114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
01178
                  110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
01179
01180
                  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}, {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
01181
01182
01183
01184
                  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,
01185
01186
                  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,
01187
01188
                  206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01189
                  279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
01191
                   305.1},
01192
                {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01193
                  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,
01194
                  108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5, 102.5, 102.5, 102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
01195
01196
                  109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
01197
01198
                  241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
01199
                  286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6}
                286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.5), (301.2, 300.3, 296.6, 295.4, 295., 294.3, 291.2, 287.4, 284.9, 284.7, 284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192, 175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1, 100.4, 99.96, 99.6, 99.37, 99.32, 99.31, 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, 200.3, 201.9, 203.3, 203.3, 203.3, 203.3, 203.3, 203.4, 203.4, 203.4, 203.3, 203.3, 203.3, 203.3, 203.4, 203.4, 203.3, 203.3, 203.3, 203.3, 203.3, 203.4, 203.4, 203.4, 203.3, 203.3, 203.3, 203.3, 203.4, 203.4, 203.4, 203.3, 203.3, 203.3, 203.3, 203.4, 203.4, 203.4, 203.4, 203.4, 203.3, 203.3, 203.3, 203.3, 203.3, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4, 203.4
01200
01201
01202
01204
01205
01206
                  280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
01207
                  281.7, 281.1, 281.2}
01208
01210
                double doy, p0, p1, pt;
01211
01212
                int imon, ilat;
01213
                /* Get day of year... */
doy = fmod(t / 86400., 365.25);
01214
01215
                while (doy < 0)
01216
                    doy += 365.25;
01217
01218
                /* Get indices... */
01219
               imon = locate(doys, 12, doy);
ilat = locate(lats, 73, lat);
01220
01221
01223
                 /* Get tropopause pressure... */
01224
               p0 = LIN(lats[ilat], tps[imon][ilat],
01225
                                  lats[ilat + 1], tps[imon][ilat + 1], lat);
               01226
01227
                pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01228
01229
01230
                /* Return tropopause pressure... */
01231
                return pt;
01232 }
```

Here is the call graph for this function:



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

Write atmospheric data.

Definition at line 1236 of file libtrac.c.

```
01240
01241
01242
         FILE *in, *out;
01243
01244
         char line[LEN];
01245
01246
         double r, t0, t1;
01247
01248
         int ip, iq, year, mon, day, hour, min, sec;
01249
         /* Set time interval for output... */
01250
         t0 = t - 0.5 * ctl->dt_mod;
01251
01252
         t1 = t + 0.5 * ctl -> dt_mod;
01253
         /* Check if gnuplot output is requested... */ if (ctl->atm_gpfile[0] != '-') {
01254
01255
01256
01257
            /* Write info... */
01258
           printf("Plot atmospheric data: %s.png\n", filename);
01259
01260
            /* Create gnuplot pipe... */
           if (!(out = popen("gnuplot", "w")))
    ERRMSG("Cannot create pipe to gnuplot!");
01261
01262
01263
           /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
01264
01265
01266
01267
            /\star Set time string... \star/
           jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01268
01269
                     year, mon, day, hour, min);
01271
01272
            /* Dump gnuplot file to pipe... */
           if (!(in = fopen(ctl->atm_gpfile, "r")))
ERRMSG("Cannot open file!");
01273
01274
           while (fgets(line, LEN, in))
fprintf(out, "%s", line);
01275
01276
01277
           fclose(in);
01278
01279
01280
         else {
01281
01282
            /* Write info... */
           printf("Write atmospheric data: %s\n", filename);
01283
01284
01285
            /* Create file... */
           if (!(out = fopen(filename, "w")))
01286
              ERRMSG("Cannot create file!");
01287
01288
01289
01290
          /* Write header... */
01291
         fprintf(out,
01292
                    "# $1 = time [s]\n"
                   "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01293
01294
         for (iq = 0; iq < ctl->nq; iq+1)
fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
01295
01296
```

```
01297
                ctl->qnt_unit[iq]);
01298
       fprintf(out, "\n");
01299
       /* Write data... */
for (ip = 0; ip < atm->np; ip++) {
01300
01301
01302
01303
         /* Check time... */
01304
         if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
01305
01306
        01307
01308
01309
01310
01311
01312
           fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01313
01314
         fprintf(out, "\n");
01315
01316
01317
       /* Close file... */
01318
       fclose(out);
01319 }
```

Here is the call graph for this function:



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

Write CSI data.

Definition at line 1323 of file libtrac.c.

```
01327
01328
01329
        static FILE *in, *out;
01330
01331
        static char line[LEN];
01332
01333
        static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01334
          rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01335
01336
        static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01337
01338
         /* Init... */
01339
         if (!init) {
01340
          init = 1;
01341
          /* Check quantity index for mass... */
if (ctl->qnt_m < 0)</pre>
01342
01343
01344
             ERRMSG("Need quantity mass to analyze CSI!");
01345
01346
           /* Open observation data file... */
           printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
if (!(in = fopen(ctl->csi_obsfile, "r")))
01347
01348
01349
             ERRMSG("Cannot open file!");
01350
01351
           /* Create new file... */
           printf("Write CSI data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01352
01353
01354
             ERRMSG("Cannot create file!");
01355
01356
           /* Write header... */
```

```
01357
         fprintf(out,
01358
                  "# $1 = time [s] \n"
01359
                  "# $2 = number of hits (cx) n"
                  "# $3 = number of misses (cy) \n"
01360
                   "# $4 = number of false alarms (cz)\n"
01361
                  "# $5 = number of observations (cx + cy)\n"
"# $6 = number of forecasts (cx + cz)\n"
01362
01363
01364
                  "# $7 = bias (forecasts/observations)
                                                          [%%]\n"
                  "# $8 = probability of detection (POD) [%%]\n" # $9 = false alarm rate (FAR) [%%]\n"
01365
01366
                  "# $10 = critical success index (CSI) [%%]\n\n");
01367
01368
01369
        /* Set time interval... */
01370
01371
        t0 = t - 0.5 * ctl->dt_mod;
        t1 = t + 0.5 * ct1->dt_mod;
01372
01373
01374
        /* Initialize grid cells... */
01375
        for (ix = 0; ix < ctl->csi_nx; ix++)
         for (iy = 0; iy < ctl->csi_ny; iy++)
01376
01377
            for (iz = 0; iz < ctl->csi_nz; iz++)
01378
              modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01379
01380
        /* Read data... */
01381
        while (fgets(line, LEN, in)) {
01382
          /* Read data... */
01383
          if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01384
01385
              5)
01386
            continue:
01387
01388
          /* Check time... */
01389
          <u>if</u> (rt < t0)
01390
            continue;
          if (rt > t1)
01391
01392
            break:
01393
01394
          /* Calculate indices... */
01395
          ix = (int) ((rlon - ctl->csi_lon0))
01396
                       / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01397
          iy = (int) ((rlat - ctl->csi_lat0))
                       / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01398
          iz = (int) ((rz - ctl->csi_z0)
01399
01400
                       / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01401
          /* Check indices... */
01402
01403
          if (ix < 0 || ix >= ctl->csi_nx ||
              iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01404
01405
            continue:
01406
01407
          /* Get mean observation index... */
01408
          obsmean[ix][iy][iz] += robs;
01409
          obscount[ix][iy][iz]++;
01410
01411
01412
        /* Analyze model data... */
        for (ip = 0; ip < atm->np; ip++) {
01414
01415
          /* Check time... */
01416
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
01417
            continue:
01418
01419
          /* Get indices... */
          ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01420
01421
                       / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01422
          iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
          01423
01424
01425
01426
01427
          /* Check indices... */
          if (ix < 0 || ix >= ctl->csi_nx ||
01428
              iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01429
01430
            continue:
01431
01432
          /* Get total mass in grid cell... */
01433
          modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01434
01435
01436
        /* Analyze all grid cells... */
        for (ix = 0; ix < ctl->csi_nx; ix++)
  for (iy = 0; iy < ctl->csi_ny; iy++)
01437
01438
            for (iz = 0; iz < ctl->csi_nz; iz++) {
01439
01440
01441
               /* Calculate mean observation index... */
              if (obscount[ix][iy][iz] > 0)
01442
01443
                obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
```

```
01444
                 /* Calculate column density... */
01445
                 /* Calculate Column Gensity... */
if (modmean[ix][iy][iz] > 0) {
  dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
  dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
  lat = ctl->csi_lat0 + dlat * (iy + 0.5);
01446
01447
01448
01449
                   area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180. * cos(lat * M_PI / 180.);
01450
01451
01452
                   modmean[ix][iy][iz] /= (1e6 * area);
01453
01454
                 /* Calculate CSI... */
01455
01456
                 if (obscount[ix][iy][iz] > 0) {
01457
                   if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01458
                        modmean[ix][iy][iz] >= ctl->csi_modmin)
01459
                   else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01460
                               modmean[ix][iy][iz] < ctl->csi_modmin)
01461
01462
                      cy++;
01463
                   else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
01464
                               modmean[ix][iy][iz] >= ctl->csi_modmin)
01465
                      cz++;
01466
                }
01467
01468
01469
         /* Write output... */
01470
         if (fmod(t, ctl->csi_dt_out) == 0) {
01471
           /* Write... */ fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
01472
01473
01474
                      t, cx, cy, cz, cx + cy, cx + cz, (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
01475
                      (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
(cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
01476
01477
01478
                       (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01479
01480
            /* Set counters to zero... */
            cx = cy = cz = 0;
01481
01482
01483
01484
         /* Close file... */
         if (t == ctl->t_stop)
01485
01486
           fclose(out);
01487 }
```

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

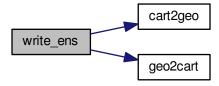
Write ensemble data.

Definition at line 1491 of file libtrac.c.

```
01495
                  {
01496
01497
       static FILE *out;
01498
01499
       static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
01500
        t0, t1, x[NENS][3], xm[3];
01501
01502
       static int init, ip, iq;
01503
01504
       static size_t i, n;
01505
01506
       /* Init... */
01507
       if (!init) {
01508
         init = 1:
01509
          /* Check quantities... */
01510
01511
          if (ctl->qnt_ens < 0)
01512
          ERRMSG("Missing ensemble IDs!");
01513
01514
          /* Create new file... */
          printf("Write ensemble data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01515
01516
01517
            ERRMSG("Cannot create file!");
01518
          /* Write header... */
01519
          01520
01521
01522
                  "# $2 = altitude [km] \n"
01523
                  "# $3 = longitude [deg]\n" "# <math>$4 = latitude [deg]\n");
```

```
01525
01526
            01527
01528
01529
01530
01531
01532
         /* Set time interval... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01533
01534
01535
01536
01537
          /* Init... */
01538
         ens = GSL_NAN;
01539
         n = 0;
01540
         /* Loop over air parcels... */
for (ip = 0; ip < atm->np; ip++) {
01541
01542
01544
            /* Check time... */
01545
            if (atm->time[ip] < t0 || atm->time[ip] > t1)
01546
             continue;
01547
01548
            /* Check ensemble id... */
            if (atm->q[ctl->qnt_ens][ip] != ens) {
01549
01550
01551
               /* Write results... */
01552
              if (n > 0) {
01553
                 /* Get mean position... */
xm[0] = xm[1] = xm[2] = 0;
for (i = 0; i < n; i++) {
01554
01555
01556
                   xm[0] += x[i][0] / (double) n;
xm[1] += x[i][1] / (double) n;
xm[2] += x[i][2] / (double) n;
01557
01558
01559
01560
                 cart2geo(xm, &dummy, &lon, &lat);
fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
01561
01562
01563
                          lat);
01564
01565
                 /* Get quantity statistics... */
                 for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01566
01567
01568
                   fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01569
                 for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01570
01571
                   fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01572
01573
01574
                 fprintf(out, " %lu\n", n);
01575
01576
01577
              /* Init new ensemble... */
01578
              ens = atm->q[ctl->qnt_ens][ip];
01579
              n = 0;
01580
01581
01582
            /* Save data...
01583
            p[n] = atm->p[ip];
01584
            geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
            for (iq = 0; iq < ctl->nq; iq++)
q[iq][n] = atm->q[iq][ip];
if ((++n) >= NENS)
01585
01586
01587
01588
              ERRMSG("Too many data points!");
01589
01590
         /* Write results... */
01591
01592
         if (n > 0) {
01593
01594
            /* Get mean position... */
            for (i = 0; i < n; i++) {
    xm[0] += x[i][0] / (double) n;
    xm[1] += x[i][1] / (double) n;</pre>
01595
01596
01597
01598
01599
              xm[2] += x[i][2] / (double) n;
01600
            cart2geo(xm, &dummy, &lon, &lat);
fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
01601
01602
01603
            /* Get quantity statistics... */
01604
            for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01605
01606
01607
01608
            for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01609
01610
```

Here is the call graph for this function:



5.11.2.31 void write\_grid ( const char \* filename, ctl\_t \* ctl, met\_t \* met0, met\_t \* met1, atm\_t \* atm, double t )

Write gridded data.

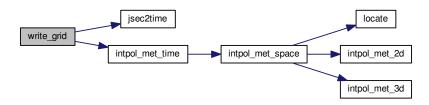
Definition at line 1623 of file libtrac.c.

```
01629
                     {
01630
         FILE *in, *out;
01631
01632
01633
         char line[LEN];
01634
         static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01635
           area, rho_air, press, temp, cd, mmr, t0, t1, r;
01636
01637
01638
         static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01639
         /* Check dimensions... */
if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
    ERRMSG("Grid dimensions too large!");
01640
01641
01642
01643
         /* Check quantity index for mass... */ if (ctl->qnt_m < 0)
01644
01645
           ERRMSG("Need quantity mass to write grid data!");
01646
01647
         /* Set time interval for output... */
01648
         t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01649
01650
01651
01652
         /\star Set grid box size... \star/
         dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
01653
01654
         dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
01655
01656
01657
          /* Initialize grid... */
         for (ix = 0; ix < ctl->grid_nx; ix++)
01658
          for (iy = 0; iy < ctl->grid_ny; iy++)
    for (iz = 0; iz < ctl->grid_nz; iz++)
01659
01660
01661
                grid_m[ix][iy][iz] = 0;
01662
01663
          /* Average data... */
01664
         for (ip = 0; ip < atm->np; ip++)
           if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {</pre>
01665
01666
01667
              /* Get index... */
01668
              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);
01670
01671
              /* Check indices... */
if (ix < 0 || ix >= ctl->grid_nx ||
    iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01672
01673
01674
01675
01676
              /* Add mass... */
grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01677
01678
01679
01680
         /* Check if gnuplot output is requested... */
if (ctl->grid_gpfile[0] != '-') {
01681
01682
01683
01684
            /* Write info... */
            printf("Plot grid data: %s.png\n", filename);
01685
01686
01687
            /* Create gnuplot pipe... */
            if (!(out = popen("gnuplot", "w")))
01688
01689
              ERRMSG("Cannot create pipe to gnuplot!");
01690
            /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
01691
01692
01693
01694
            /* Set time string... */
            jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01695
01696
01697
                      year, mon, day, hour, min);
01698
            /* Dump gnuplot file to pipe... */
if (!(in = fopen(ctl->grid_gpfile, "r")))
    ERRMSG("Cannot open file!");
01699
01700
01701
            while (fgets(line, LEN, in))
fprintf(out, "%s", line);
01702
01703
01704
            fclose(in);
01705
01706
01707
         else {
01708
01709
            /* Write info... */
           printf("Write grid data: %s\n", filename);
01710
01711
01712
            /* Create file... */
01713
           if (!(out = fopen(filename, "w")))
01714
               ERRMSG("Cannot create file!");
01715
01716
         /* Write header... */
01717
01718
         fprintf(out,
                    "# $1 = time [s]\n"
01720
                    "# $2 = altitude [km] \n"
01721
                    "# $3 = longitude [deg] \n"
                    "# $4 = latitude [deg]\n"
01722
                    "# $5 = surface area [km^2]\n"
01723
01724
                    "# $6 = layer width [km] \n"
01725
                    "# $7 = temperature [K]\n"
01726
                    "# $8 = \text{column density } [kg/m^2] \n"
01727
                    "# $9 = mass mixing ratio [1]\n\n");
01728
01729
         /* Write data... */
         /* white data... */
for (ix = 0; ix < ctl->grid_nx; ix++) {
    if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01730
01732
               fprintf(out, "\n");
01733
                 (iy = 0; iy < ctl->grid_ny; iy++) {
01734
              if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
               fprintf(out, "\n");
for (iz = 0; iz < ctl->grid_nz; iz++)
01735
01736
01737
                 if (!ctl->grid_sparse
                      || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01739
01740
                    /* Set coordinates... */
                    z = ctl->grid_z0 + dz * (iz + 0.5);
lon = ctl->grid_lon0 + dlon * (ix + 0.5);
01741
01742
                    lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01743
01744
01745
                    /* Get pressure and temperature... */
01746
                    press = P(z);
                    intpol_met_time(met0, met1, t, press, lon, lat,
    NULL, &temp, NULL, NULL, NULL, NULL, NULL);
01747
01748
01749
01750
                    /* Calculate surface area... */
                   area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
 * cos(lat * M_PI / 180.);
01751
01752
01753
                   /* Calculate column density... */
cd = grid_m[ix][iy][iz] / (le6 * area);
01754
01755
```

```
/* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01757
01758
01759
01760
                 01761
01762
01763
01764
01765
           }
        }
01766
01767
01768
        /* Close file... */
01769
        fclose(out);
01770 }
```

Here is the call graph for this function:



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

Write profile data.

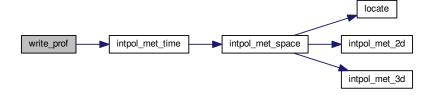
Definition at line 1774 of file libtrac.c.

```
01780
                      {
01781
01782
         static FILE *in, *out;
01783
01784
         static char line[LEN];
01785
01786
         static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
           rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, mmr, h2o, o3;
01787
01788
01789
01790
         static int init, obscount[GX][GY], ip, ix, iy, iz;
01791
          /* Init... */
01792
          if (!init) {
01793
01794
            init = 1;
01795
01796
            /\star Check quantity index for mass... \star/
01797
            if (ctl->qnt_m < 0)
01798
              ERRMSG("Need quantity mass!");
01799
            /* Check dimensions... */    if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
01800
01801
              ERRMSG("Grid dimensions too large!");
01802
01803
01804
            /\star Open observation data file... \star/
            \label{lem:printf}  \mbox{"Read profile observation data: $s\n", ctl->prof_obsfile);} 
01805
            if (!(in = fopen(ctl->prof_obsfile, "r")))
    ERRMSG("Cannot open file!");
01806
01807
01808
01809
            /* Create new file... */
            printf("Write profile data: %s\n", filename);
if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");
01810
01811
01812
01813
01814
            /* Write header... */
01815
            fprintf(out,
```

```
"# $1 = time [s] \n"
                      "# $2 = altitude [km] \n"
01817
01818
                      "# $3 = longitude [deg] \n"
                      "# $4 = latitude [deg]\n"
01819
                      "# S5
                              = pressure [hPa]\n"
01820
                      "# $6 = temperature [K]\n"
01821
                              = mass mixing ratio [1]\n"
01822
01823
                      "# $8
                              = H2O volume mixing ratio [1]\n"
01824
                      "# $9 = 03 volume mixing ratio [1]\n"
01825
                      "# $10 = mean BT index [K]\n");
01826
01827
            /* 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;
01828
01829
01830
01831
01832
01833
          /* Set time interval... */
         t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01834
01835
01836
         /* Initialize... */
for (ix = 0; ix < ctl->prof_nx; ix++)
   for (iy = 0; iy < ctl->prof_ny; iy++) {
    obsmean[ix][iy] = 0;
01837
01838
01839
01840
               obscount[ix][iy] = 0;
01841
01842
               tmean[ix][iy] = 0;
01843
               for (iz = 0; iz < ctl->prof_nz; iz++)
01844
                 mass[ix][iy][iz] = 0;
01845
01846
01847
         /* Read data... */
01848
         while (fgets(line, LEN, in)) {
01849
            /* Read data... */
if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01850
01851
01852
             continue;
01853
01854
            /* Check time... */
01855
            if (rt < t0)</pre>
01856
            continue;
if (rt > t1)
01857
01858
              break:
01859
01860
            /* Calculate indices... */
            ix = (int) ((rlon - ctl->prof_lon0) / dlon);
iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01861
01862
01863
            /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01864
01865
01866
             continue;
01867
01868
            /\star Get mean observation index... \star/
            obsmean[ix][iy] += robs;
tmean[ix][iy] += rt;
01869
01870
01871
            obscount[ix][iy]++;
01872
01873
01874
          /* Analyze model data... */
01875
          for (ip = 0; ip < atm->np; ip++) {
01876
01877
            /* Check time... */
01878
            if (atm->time[ip] < t0 || atm->time[ip] > t1)
01879
              continue;
01880
            /* Get indices... */
01881
           ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
01882
01883
            iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01884
01886
            /* Check indices... */
            if (ix < 0 || ix >= ctl->prof_nx ||
01887
                iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01888
01889
               continue:
01890
01891
            /* Get total mass in grid cell... */
01892
            mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01893
01894
01895
          /* Extract profiles... */
         for (ix = 0; ix < ctl->prof_nx; ix++)
  for (iy = 0; iy < ctl->prof_ny; iy++)
01896
01897
01898
              if (obscount[ix][iy] > 0) {
01899
                 /* Write output... */
fprintf(out, "\n");
01900
01901
01902
```

```
/* Loop over altitudes... */
01904
              for (iz = 0; iz < ctl->prof_nz; iz++) {
01905
                /* Set coordinates... */
01906
                z = ctl->prof_z0 + dz * (iz + 0.5);
lon = ctl->prof_lon0 + dlon * (ix + 0.5);
01907
01908
01909
                lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01910
01911
                /\star Get meteorological data... \star/
01912
                press = P(z);
                01913
01914
01915
01916
               /* Calculate mass mixing ratio... */
               rho_air = 100. * press / (287.058 * temp);
area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
* cos(lat * M_PI / 180.);
01917
01918
01919
01920
               mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
01921
               01923
01924
                        z, lon, lat, press, temp, mmr, h2o, o3,
01925
01926
                        obsmean[ix][iy] / obscount[ix][iy]);
01927
01928
01929
01930
        /* Close file... */
        if (t == ctl->t_stop)
01931
01932
          fclose(out);
01933 }
```

Here is the call graph for this function:



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

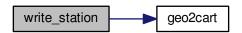
Write station data.

Definition at line 1937 of file libtrac.c.

```
{
01942
01943
       static FILE *out;
01944
       static double rmax2, t0, t1, x0[3], x1[3];
01945
01946
01947
       static int init, ip, iq;
01948
01949
        /* Init... */
       if (!init) {
01950
01951
         init = 1;
01952
01953
         /* Write info... */
01954
         printf("Write station data: %s\n", filename);
01955
01956
          /* Create new file... */
         if (!(out = fopen(filename, "w")))
01957
01958
           ERRMSG("Cannot create file!");
01959
01960
          /* Write header... */
```

```
01961
          fprintf(out,
01962
                  "# $1 = time [s] \n"
                  "# \$2 = altitude [km]\n"
"# \$3 = longitude [deg]\n" "# \$4 = latitude [deg]\n");
01963
01964
          for (iq = 0; iq < ctl->nq; iq++)
  fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
01965
          01966
01967
01968
01969
01970
          /\star Set geolocation and search radius... \star/
          geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
rmax2 = gsl_pow_2(ctl->stat_r);
01971
01972
01973
01974
01975
        /\star Set time interval for output... \star/
       t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01976
01977
01978
01979
        /* Loop over air parcels... */
01980
        for (ip = 0; ip < atm->np; ip++) {
01981
          01982
01983
01984
            continue;
01985
01986
          /* Check station flag... */
01987
          if (ctl->qnt_stat >= 0)
01988
           if (atm->q[ctl->qnt_stat][ip])
01989
              continue;
01990
01991
          /* Get Cartesian coordinates... */
01992
          geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
01993
01994
          /\star Check horizontal distance... \star/
01995
          if (DIST2(x0, x1) > rmax2)
01996
            continue;
01997
01998
          /* Set station flag... */
01999
          if (ctl->qnt_stat >= 0)
02000
           atm->q[ctl->qnt_stat][ip] = 1;
02001
          02002
02003
02004
          for (iq = 0; iq < ctl->nq; iq++) {
   fprintf(out, " ");
02005
02006
02007
            fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02008
02009
          fprintf(out, "\n");
02010
02011
02012
        /* Close file... */
02013
        if (t == ctl->t\_stop)
02014
          fclose(out);
02015 }
```

Here is the call graph for this function:



```
00001 /\star
00002 This file is part of MPTRAC.
00003
00004 MPTRAC is free software: you can redistribute it and/or modify
00005 it under the terms of the GNU General Public License as published by
```

```
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
      Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00028
00029 void cart2geo(
00030
      double *x,
00031
      double *z,
00032
      double *lon,
00033
      double *lat) {
00034
00035
      double radius;
00036
00037
      radius = NORM(x);
00038
      *lat = asin(x[2] / radius) * 180 / M_PI;
00039
      *lon = atan2(x[1], x[0]) * 180 / M_PI;
00040
      *z = radius - RE;
00041 }
00042
00044
00045 double deg2dx(
00046
      double dlon,
00047
      double lat) {
00048
00049
      return dlon * M_PI * RE / 180. * cos(lat / 180. * M_PI);
00050 }
00051
00053
00054 double deg2dy(
00055
      double dlat) {
00056
00057
      return dlat * M_PI * RE / 180.;
00058 }
00059
00061
00062 double dp2dz(
00063
      double dp,
00064
      double p) {
00065
00066
      return -dp * H0 / p;
00067 }
00068
00070
00071 double dx2deg(
00072
      double dx,
00073
      double lat) {
00074
00075
      /\star Avoid singularity at poles... \star/
00076
      if (lat < -89.999 || lat > 89.999)
00077
       return 0;
00078
      else
00079
       return dx * 180. / (M_PI * RE * cos(lat / 180. * M_PI));
00080 }
00081
00083
00084 double dy2deg(
00085
      double dy) {
00086
00087
      return dy * 180. / (M_PI * RE);
00088 }
00089
00091
00092 double dz2dp(
00093
      double dz,
00094
      double p) {
00095
00096
      return -dz * p / H0;
00097 }
```

```
00100
00101 void geo2cart(
00102
       double z,
00103
       double lon,
00104
       double lat,
00105
       double *x) {
00106
00107
       double radius;
00108
00109
       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);
00110
00111
00112
00113 }
00114
00116
00117 void get_met(
      ctl_t * ctl,
char *metbase,
00118
00119
00120
       double t,
      met_t * met0,
met_t * met1) {
00121
00122
00123
00124
       char filename[LEN];
00125
00126
       static int init;
00127
00128
       /* Init... */
00129
       if (!init) {
00130
        init = 1;
00131
         get_met_help(t, -1, metbase, ctl->dt_met, filename);
read_met(ctl, filename, met0);
00132
00133
00134
00135
         get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
     dt_met, filename);
00136
        read_met(ctl, filename, met1);
00137
00138
00139
       /* Read new data for forward trajectories... */
00140
       if (t > met1->time && ctl->direction == 1) {
       memcpy(met0, met1, sizeof(met_t));
00141
00142
         get_met_help(t, 1, metbase, ctl->dt_met, filename);
00143
        read_met(ctl, filename, met1);
00144
00145
00146
       /* Read new data for backward trajectories... */
       if (t < met0->time && ctl->direction == -1) {
00147
00148
        memcpy(met1, met0, sizeof(met_t));
00149
         get_met_help(t, -1, metbase, ctl->dt_met, filename);
00150
        read_met(ctl, filename, met0);
00151
00152 }
00155
00156 void get_met_help(
00157
       double t.
00158
       int direct,
00159
       char *metbase,
00160
       double dt_met,
00161
       char *filename) {
00162
00163
       double t6, r;
00164
00165
       int year, mon, day, hour, min, sec;
00166
00167
       /\star Round time to fixed intervals... \star/
00168
       if (direct == -1)
        t6 = floor(t / dt_met) * dt_met;
00169
00170
       else
00171
        t6 = ceil(t / dt_met) * dt_met;
00172
00173
       /* Decode time... */
00174
       jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00175
00176
       /* Set filename... */
00177
       sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", metbase, year, mon, day, hour);
00178 }
00179
00181
00182 void intpol_met_2d(
00183 double array[EX][EY],
```

```
00184
         int ix,
00185
         int iy,
00186
         double wx,
00187
         double wy,
00188
         double *var) {
00189
00190
         double aux00, aux01, aux10, aux11;
00191
00192
         /* Set variables...
         aux00 = array[ix][iy];
aux01 = array[ix][iy + 1];
aux10 = array[ix + 1][iy];
00193
00194
00195
00196
         aux11 = array[ix + 1][iy + 1];
00197
00198
         /* Interpolate horizontally... */
         aux00 = wy * (aux00 - aux01) + aux01;

aux11 = wy * (aux10 - aux11) + aux11;
00199
00200
00201
         *var = wx * (aux00 - aux11) + aux11;
00202 }
00203
00205
00206 void intpol_met_3d(
00207
        float array[EX][EY][EP],
00208
         int ip,
         int ix,
00210
         int iy,
00211
         double wp,
00212
         double wx,
00213
         double wy,
00214
         double *var) {
00215
00216
         double aux00, aux01, aux10, aux11;
00217
00218
         /* Interpolate vertically...
         aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00219
         + array[ix][iy][ip + 1];

aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])

+ array[ix][iy + 1][ip + 1];
00220
00222
00223
         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])
+ array[ix + 1][iy + 1][ip + 1];
00224
00225
00226
00227
         /* Interpolate horizontally... */
aux00 = wy * (aux00 - aux01) + aux01;
aux11 = wy * (aux10 - aux11) + aux11;
00228
00229
00230
         *var = wx * (aux00 - aux11) + aux11;
00231
00232 }
00233
00235
00236 void intpol_met_space(
         met_t * met,
double p,
00237
00238
00239
         double lon,
         double lat,
00240
00241
         double *ps,
00242
         double *t,
         double *u,
00243
00244
         double *v.
00245
         double *w,
00246
         double *h2o,
00247
         double *o3) {
00248
00249
         double wp, wx, wy;
00250
00251
         int ip, ix, iv;
00252
         /* Check longitude... */
if (met->lon[met->nx - 1] > 180 && lon < 0)
00253
00254
00255
           lon += 360;
00256
00257
         /* Get indices... */
         ip = locate(met->p, met->np, p);
ix = locate(met->lon, met->nx, lon);
00258
00259
00260
         iy = locate(met->lat, met->ny, lat);
00261
         /* Get weights... */
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]);
00262
00263
00264
00265
00266
00267
         /* Interpolate..
         if (ps != NULL)
  intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
if (t != NULL)
00268
00269
00270
```

```
00271
         intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
00272
       if (u != NULL)
00273
        intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
       if (v != NULL)
00274
        intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
00275
00276
       if (w != NULL)
00277
        intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
00278
       if (h2o != NULL)
00279
        intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
00280
       if (o3 != NULL)
00281
        intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00282 }
00283
00285
00286 void intpol_met_time(
00287
      met_t * met0,
met_t * met1,
00288
00289
       double ts,
00290
       double p,
00291
       double lon,
00292
       double lat,
00293
       double *ps,
00294
       double *t.
00295
       double *u,
00296
       double *v,
00297
       double *w,
00298
       double *h2o,
00299
      double *o3) {
00300
00301
       double h2o0, h2o1, o30, o31, ps0, ps1, t0, t1, u0, u1, v0, v1, w0, w1, wt;
00302
00303
       /* Spatial interpolation... */
       00304
00305
00306
00307
                       u == NULL ? NULL : &u0,
                       v == NULL ? NULL : &v0,
00308
                       w == NULL ? NULL : &w0,
00309
00310
                       h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
       00311
00312
00313
00314
                       u == NULL ? NULL : &u1,
                       v == NULL ? NULL : &v1,
00315
00316
                       w == NULL ? NULL : &w1,
00317
                       h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00318
       /\star Get weighting factor... \star/
00319
      wt = (met1->time - ts) / (met1->time - met0->time);
00320
00321
00322
       /* Interpolate... */
00323
       if (ps != NULL)
       *ps = wt * (ps0 - ps1) + ps1;
if (t != NULL)
00324
00325
00326
        *t = wt * (t0 - t1) + t1;
       if (u != NULL)
00327
00328
        *u = wt * (u0 - u1) + u1;
00329
       if (v != NULL)
00330
         *v = wt * (v0 - v1) + v1;
       if (w != NULL)
00331
        *w = wt * (w0 - w1) + w1;
00332
00333
       if (h2o != NULL)
00334
        *h2o = wt * (h2o0 - h2o1) + h2o1;
       if (o3 != NULL)
00335
00336
        *o3 = wt * (o30 - o31) + o31;
00337 }
00338
00340
00341 void jsec2time(
00342
      double jsec,
00343
       int *year,
00344
       int *mon.
00345
       int *day,
00346
       int *hour,
00347
       int *min,
00348
       int *sec,
00349
       double *remain) {
00350
00351
      struct tm t0, *t1;
00352
00353
       time_t jsec0;
00354
00355
      t0.tm_year = 100;
       t0.tm_mon = 0;
00356
       t0.tm_mday = 1;
00357
```

```
00358
        t0.tm\_hour = 0;
00359
        t0.tm_min = 0;
00360
        t0.tm_sec = 0;
00361
00362
         jsec0 = (time_t) jsec + timegm(&t0);
        t1 = gmtime(&jsec0);
00363
00364
00365
        *year = t1->tm_year + 1900;
        *mon = t1->tm_mon + 1;
*day = t1->tm_mday;
00366
00367
        *hour = t1->tm_hour;
00368
00369
        *min = t1->tm_min;
        *sec = t1->tm_sec;
00370
00371
        *remain = jsec - floor(jsec);
00372 }
00373
00375
00376 int locate(
00377
        double *xx,
00378
        int n,
00379
        double x) {
00380
00381
        int i, ilo, ihi;
00382
00383
        ilo = 0;
        ihi = n - 1;
00384
        i = (ihi + ilo) >> 1;
00385
00386
00387
        if (xx[i] < xx[i + 1])
         while (ihi > ilo + 1) {
  i = (ihi + ilo) >> 1;
00388
00389
00390
             if (xx[i] > x)
00391
               ihi = i;
00392
             else
               ilo = i;
00393
00394
        } else
          while (ihi > ilo + 1) {
00396
            i = (ihi + ilo) >> 1;
00397
             if (xx[i] <= x)</pre>
00398
               ihi = i;
             else
00399
00400
               ilo = i:
00401
          }
00402
00403
        return ilo;
00404 }
00405
00407
00408 void read_atm(
00409
       const char *filename,
00410
        ctl_t * ctl,
00411
        atm_t * atm) {
00412
00413
        FILE *in;
00415
        char line[LEN], *tok;
00416
00417
        int iq;
00418
        /* Init... */
atm->np = 0;
00419
00420
00421
00422
        /* Write info... */
00423
        printf("Read atmospheric data: %s\n", filename);
00424
00425
        /* Open file... */
        if (!(in = fopen(filename, "r")))
00426
          ERRMSG("Cannot open file!");
00427
00428
00429
        /* Read line... */
00430
        while (fgets(line, LEN, in)) {
00431
          /* 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]);
00432
00433
00434
00435
00436
00437
00438
00439
00440
           /* Convert altitude to pressure... */
00441
           atm \rightarrow p[atm \rightarrow np] = P(atm \rightarrow p[atm \rightarrow np]);
00442
           /* Increment data point counter... */
if ((++atm->np) > NP)
00443
00444
```

```
00445
            ERRMSG("Too many data points!");
00446
00447
00448
       /* Close file... */
00449
       fclose(in);
00450
00451
        /* Check number of points... */
00452
        if (atm->np < 1)
00453
          ERRMSG("Can not read any data!");
00454 }
00455
00457
00458 void read_ctl(
00459
        const char *filename,
00460
        int argc,
       char *argv[],
ctl_t * ctl) {
00461
00462
00463
00464
        int ip, iq;
00465
00466
        /* Write info... */
        00467
00468
00469
                argv[0], __DATE__, __TIME__);
00470
00471
        /* Initialize quantity indices... */
        ctl->qnt_ens = -1;
ctl->qnt_m = -1;
00472
00473
        ctl \rightarrow qnt_r = -1;
00474
00475
        ctl->gnt rho = -1;
00476
        ctl->qnt_ps = -1;
00477
        ctl->qnt_p = -1;
00478
        ct1->qnt_t = -1;
        ct1->qnt_u = -1;
00479
        ctl \rightarrow qnt_v = -1;
00480
        ctl->qnt_w = -1;
00481
        ctl->qnt_h2o = -1;
00482
00483
        ctl->qnt_o3 = -1;
00484
        ctl->qnt\_theta = -1;
00485
        ctl->qnt_pv = -1;
00486
        ctl->qnt\_tice = -1;
        ctl->qnt\_tsts = -1;
00487
00488
       ctl->qnt_tnat = -1;
00489
       ctl->qnt\_stat = -1;
00490
00491
        /* Read quantities... */
        ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
00492
        for (iq = 0; iq < ctl->nq; iq++) {
00493
00494
          /* 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",
00495
00496
00497
00498
                    ctl->qnt_format[iq]);
00499
          /* Try to identify quantity... */
if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
00500
00501
00502
            ctl->qnt_ens = iq;
00503
            sprintf(ctl->qnt_unit[iq], "-");
          } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
  ctl->qnt_m = iq;
00504
00505
            sprintf(ctl->qnt_unit[iq], "kg");
00506
          } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
ctl->qnt_r = iq;
00507
00508
00509
            sprintf(ctl->qnt_unit[iq], "m");
00510
          } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
00511
            ctl->qnt_rho = iq;
            sprintf(ctl->qnt_unit[iq], "kg/m^3");
00512
          } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
00513
            ctl->qnt_ps = iq;
00515
            sprintf(ctl->qnt_unit[iq], "hPa");
00516
          } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
            ctl->qnt_p = iq;
sprintf(ctl->qnt_unit[iq], "hPa");
00517
00518
          } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
  ctl->qnt_t = iq;
00519
00520
00521
            sprintf(ctl->qnt_unit[iq], "K");
00522
          } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
            ctl->qnt_u = iq;
00523
            sprintf(ctl->qnt_unit[iq], "m/s");
00524
          } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
00525
            ctl->qnt_v = iq;
00527
            sprintf(ctl->qnt_unit[iq], "m/s");
          } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
00528
            ctl->qnt_w = iq;
00529
          sprintf(ctl->qnt_unit[iq], "hPa/s");
} else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
00530
00531
```

```
ctl->qnt_h2o = iq;
             sprintf(ctl->qnt_unit[iq], "1");
00533
00534
           } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
             ctl->qnt_o3 = iq;
00535
00536
             sprintf(ctl->qnt_unit[iq], "1");
           } else if (stromp(ctl->qnt_name[iq], "theta") == 0) {
ctl->qnt_theta = iq;
00537
00539
             sprintf(ctl->qnt_unit[iq], "K");
           } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
00540
00541
            ctl->qnt_pv = iq;
            sprintf(ctl->qnt_unit[iq], "PVU");
00542
          } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
ctl->qnt_tice = iq;
00543
00544
             sprintf(ctl->qnt_unit[iq], "K");
00545
          } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
00546
            ctl->qnt_tsts = iq;
00547
             sprintf(ctl->qnt_unit[iq], "K");
00548
          } else if (strcmp(ct1->qnt_name[iq], "tnat") == 0) {
  ct1->qnt_tnat = iq;
00549
            sprintf(ctl->qnt_unit[iq], "K");
00551
00552
           } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
            ctl->qnt_stat = iq;
00553
00554
             sprintf(ctl->qnt_unit[iq], "-");
00555
          } else
00556
             scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00557
00558
00559
         /* Time steps of simulation... */
00560
        ctl->direction =
        (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
if (ctl->direction != -1 && ctl->direction != 1)
00561
00562
00563
          ERRMSG("Set DIRECTION to -1 or 1!");
00564
         ctl->t_start =
00565
          scan_ctl(filename, argc, argv, "T_START", -1, "-1e100", NULL);
        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);
00566
00567
00568
         /* Meteorological data... */
00570
        ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
00571
         ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
00572
         if (ctl->met_np > EP)
          ERRMSG("Too many levels!");
00573
00574
         for (ip = 0; ip < ctl->met np; ip++)
00575
          ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
00576
00577
         /* Isosurface parameters... */
00578
        ctl->isosurf
        = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00579
00580
00581
00582
         /* Diffusion parameters... */
00583
        ctl->turb_dx_trop
00584
          = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00585
        ctl->turb dx strat
00586
          = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00587
        ctl->turb dz trop
           = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00589
        ctl->turb dz strat
00590
           = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00591
        ctl->turb meso :
          scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00592
00593
00594
        /* Life time of particles... */
        ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL); ctl->tdec_strat =
00595
00596
00597
          scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00598
00599
        /* PSC analysis... */
00600
        ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
        ctl->psc_hno3 =
00601
00602
          scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
00603
        /* Output of atmospheric data... */
scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
00604
00605
      atm basename);
00606 scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00607
        ctl->atm_dt_out
00608
          scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00609
        ctl->atm filter
00610
           (int) scan ctl(filename, argc, argv, "ATM FILTER", -1, "0", NULL);
00611
00612
        /* Output of CSI data... */
        scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
00614 ctl->csi_dt_out =
        scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "obs.tab",
00615
00616
```

```
ctl->csi_obsfile);
         ctl->csi_obsmin =
00618
            scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00619
00620
         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);
00621
00622
00623
00624
00625
         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);
00626
00627
00628
         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);
00629
00630
00631
00632
         ctl->csi ny =
            (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
00633
00634
         /* Output of ensemble data... */
scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
00635
00636
       ens basename);
00637
00638
         /* Output of grid data... */
         scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
00639
00640
                     ctl->grid_basename);
         scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
00641
      grid_gpfile);
00642 ctl->grid_dt_out =
            scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
00643
00644
         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);
00645
00646
00647
         ctl->grid_nz =
00648
00649
            (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00650
         ctl->grid_lon0 =
00651
           scan ctl(filename, argc, argv, "GRID LONO", -1, "-180", NULL);
00652
         ctl->grid_lon1 =
00653
            scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
00654
         ctl->grid_nx =
00655
           (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00656
         ctl->grid lat0 =
           scan ctl(filename, argc, argv, "GRID LATO", -1, "-90", NULL):
00657
00658
         ctl->grid_lat1 =
00659
           scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
00660
         ctl->grid_ny =
00661
            (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
00662
00663
         /* Output of profile data... */
         scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
00664
00665
                     ctl->prof_basename);
         scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
00666
      prof_obsfile);
         ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL); ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
00667
00668
00669
         ctl->prof nz =
00670
            (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
00671
         ctl->prof lon0 =
00672
            scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
00673
         ctl->prof_lon1
00674
           scan ctl(filename, argc, argv, "PROF LON1", -1, "180", NULL);
00675
         ctl->prof nx =
00676
            (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00677
          ctl->prof lat0 =
00678
            scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
00679
         ctl->prof_lat1 =
00680
            scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
00681
         ctl->prof_ny =
00682
            (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00683
00684
          /* Output of station data... */
         scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
00685
00686
                   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);
00687
00688
00689
00690 }
00691
00693
00694 void read met (
00695
         ctl_t * ctl,
00696
         char *filename,
00697
         met_t * met) {
00698
00699
         char tstr[10];
00700
```

```
00701
          static float help[EX * EY];
00702
00703
          int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00704
00705
          size t np, nx, ny;
00706
00707
          /* Write info... */
00708
          printf("Read meteorological data: %s\n", filename);
00709
00710
          /* Get time from filename... */
00711
          sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00712
          vear = atoi(tstr);
00713
          sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00714
          mon = atoi(tstr);
00715
          sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
          day = atoi(tstr);
sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00716
00717
00718
          hour = atoi(tstr);
          time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00720
           /* Open netCDF file...
00721
00722
          NC(nc_open(filename, NC_NOWRITE, &ncid));
00723
00724
          /* Get dimensions... */
NC(nc_inq_dimid(ncid, "lon", &dimid));
NC(nc_inq_dimlen(ncid, dimid, &nx));
00725
00726
00727
              (nx > EX)
00728
            ERRMSG("Too many longitudes!");
00729
00730
          NC(nc_inq_dimid(ncid, "lat", &dimid));
00731
          NC(nc_ing_dimlen(ncid, dimid, &ny));
00732
              (ny > EY)
00733
            ERRMSG("Too many latitudes!");
00734
          NC(nc_inq_dimid(ncid, "lev", &dimid));
NC(nc_inq_dimlen(ncid, dimid, &np));
00735
00736
00737
          if (np > EP)
00738
            ERRMSG("Too many levels!");
00739
00740
          /* Store dimensions... */
          met->np = (int) np;
met->nx = (int) nx;
00741
00742
00743
          met->ny = (int) ny;
00744
          /* Get horizontal grid... */
NC(nc_inq_varid(ncid, "lon", &varid));
00745
00746
          NC(nc_get_var_double(ncid, varid, met->lon));
NC(nc_inq_varid(ncid, "lat", &varid));
00747
00748
00749
          NC(nc_get_var_double(ncid, varid, met->lat));
00750
00751
          /* Read meteorological data... */
         /* Read meteorological data... */
read_met_help(ncid, "t", "T", met, met->t, 1.0);
read_met_help(ncid, "u", "U", met, met->u, 1.0);
read_met_help(ncid, "v", "V", met, met->v, 1.0);
read_met_help(ncid, "w", "W", met, met->w, 0.01f);
read_met_help(ncid, "q", "Q", met, met->help(ncid, "q", "Q", met, met->help(ncid, "o3", "o3", met, met->o3, 0.602f);
00752
00753
00754
00755
00756
00757
00758
00759
          /* Meteo data on pressure levels... */
00760
          if (ctl->met_np <= 0) {</pre>
00761
00762
            /* Read pressure levels from file... */
NC(nc_inq_varid(ncid, "lev", &varid));
00763
00764
            NC(nc_get_var_double(ncid, varid, met->p));
00765
             for (ip = 0; ip < met->np; ip++)
00766
              met->p[ip] /= 100.;
00767
00768
             /* Extrapolate data for lower boundary... */
00769
            read met extrapolate(met):
00770
00771
00772
          /\star Meteo data on model levels... \star/
00773
          else {
00774
            /* Read pressure data from file... */
read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00775
00776
00777
00778
             /\star Interpolate from model levels to pressure levels... \star/
             read_met_ml2pl(ctl, met, met->t);
read_met_ml2pl(ctl, met, met->u);
00779
00780
00781
             read_met_ml2pl(ctl, met, met->v);
00782
             read_met_ml2pl(ctl, met, met->w);
00783
             read_met_ml2pl(ctl, met, met->h2o);
00784
             read_met_ml2pl(ctl, met, met->o3);
00785
00786
             /* Set pressure levels... */
00787
            met->np = ctl->met np;
```

```
for (ip = 0; ip < met->np; ip++)
00789
            met->p[ip] = ctl->met_p[ip];
00790
00791
00792
         /* Check ordering of pressure levels... */
00793
        for (ip = 1; ip < met->np; ip++)
  if (met->p[ip - 1] < met->p[ip])
00794
00795
             ERRMSG("Pressure levels must be descending!");
00796
        /* Read surface pressure... */
if (nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
   NC(nc_get_var_float(ncid, varid, help));
   for (iy = 0; iy < met->ny; iy++)
      for (ix = 0; ix < met->nx; ix++)
00797
00798
00799
00800
00801
        met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
} else if (nc_ing_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00802
00803
00804
          NC(nc_get_var_float(ncid, varid, help));
           for (iy = 0; iy < met->ny; iy++)
  for (ix = 0; ix < met->nx; ix++)
00805
00806
00807
               met \rightarrow ps[ix][iy] = exp(help[iy * met \rightarrow nx + ix]) / 100.;
80800
00809
           for (ix = 0; ix < met->nx; ix++)
00810
             for (iy = 0; iy < met->ny; iy++)
00811
               met->ps[ix][iy] = met->p[0];
00812
00813
         /\star Create periodic boundary conditions... \star/
        read_met_periodic(met);
00814
00815
00816
         /* Close file...
        NC(nc_close(ncid));
00817
00818 }
00819
00821
00822 void read_met_extrapolate(
00823
        met_t * met) {
00824
        int ip, ip0, ix, iy;
00826
00827
         /* Loop over columns... */
00828
        for (ix = 0; ix < met->nx; ix++)
          for (iy = 0; iy < met->ny; iy++) {
00829
00830
00831
             /* Find lowest valid data point... */
             for (ip0 = met->np - 1; ip0 >= 0; ip0--)
00832
00833
               if (!gsl_finite(met->t[ix][iy][ip0])
00834
                    || !gsl_finite(met->u[ix][iy][ip0])
00835
                    || !gsl_finite(met->v[ix][iy][ip0])
                    | !gsl_finite(met->w[ix][iy][ip0]))
00836
00837
                 break;
00838
00839
             /* Extrapolate... */
             for (ip = ip0; ip >= 0; ip--) {
00840
               met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
00841
00842
00843
               met \rightarrow v[ix][iy][ip] = met \rightarrow v[ix][iy][ip + 1];
               met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
00844
00845
               met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
00846
               met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00847
00848
           }
00849 }
00850
00852
00853 void read_met_help(
00854
        int ncid,
00855
        char *varname,
char *varname2,
00856
        met_t * met,
00858
        float dest[EX][EY][EP],
00859
        float scl) {
00860
00861
        static float help[EX * EY * EP];
00862
00863
        int ip, ix, iy, n = 0, varid;
00864
         /* Check if variable exists... */
00865
        if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00866
          if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00867
00868
             return;
00869
00870
         /* Read data... */
        NC(nc_get_var_float(ncid, varid, help));
00871
00872
00873
         /* Copy and check data... */
00874
        for (ip = 0; ip < met->np; ip++)
```

```
for (iy = 0; iy < met->ny; iy++)
            for (ix = 0; ix < met->nx; ix++) {
    dest[ix][iy][ip] = scl * help[n++];
    if (fabs(dest[ix][iy][ip] / scl) > lel4)
00876
00877
00878
00879
                  dest[ix][iy][ip] = GSL_NAN;
00880
00881 }
00882
00884
00885 void read_met_ml2pl(
        ctl_t * ctl,
met_t * met,
00886
00887
00888
        float var[EX][EY][EP]) {
00889
00890
        double aux[EP], p[EP], pt;
00891
00892
        int ip, ip2, ix, iy;
00893
00894
         /* Loop over columns... */
00895
        for (ix = 0; ix < met->nx; ix++)
00896
           for (iy = 0; iy < met->ny; iy++) {
00897
             /* Copy pressure profile... */
for (ip = 0; ip < met->np; ip++)
  p[ip] = met->pl[ix][iy][ip];
00898
00899
00900
00901
00902
              /* Interpolate... */
              for (ip = 0; ip < ctl->met_np; ip++) {
00903
               pt = ctl->met_p[ip];
if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
00904
00905
00906
                  pt = p[0];
00907
                else if ((pt > p[met->np - 1] && p[1] > p[0])
00908
                          || (pt < p[met->np - 1] && p[1] < p[0]))
00909
                  pt = p[met->np - 1];
               00910
00911
00912
00913
00914
             /* Copy data... */
for (ip = 0; ip < ctl->met_np; ip++)
    var[ix][iy][ip] = (float) aux[ip];
00915
00916
00917
00918
00919 }
00920
00922
00923 void read_met_periodic(
00924
        met_t * met) {
00925
00926
        int ip, iy;
00927
00928
         /* Check longitudes... */
         if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
00929
00930
                     + \text{ met} - > \text{lon}[1] - \text{ met} - > \text{lon}[0] - 360) < 0.01))
00931
00932
00933
         /* Increase longitude counter... */
00934
         if ((++met->nx) > EX)
          ERRMSG("Cannot create periodic boundary conditions!");
00935
00936
00937
        /* Set longitude... */
        met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
00938
      lon[0];
00939
00940
         /* Loop over latitudes and pressure levels... */
         for (iy = 0; iy < met->ny; iy++)
  for (ip = 0; ip < met->np; ip++) {
    met->ps[met->nx - 1][iy] = met->ps[0][iy];
00941
00942
00943
             met->ps[met->nx - 1][iy][ip] = met->u[0][iy][ip];
met->v[met->nx - 1][iy][ip] = met->v[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];
met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
00944
00945
00946
00947
00948
00949
             met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
00950
00951 }
00952
00954
00955 double scan_ctl(
        const char *filename,
00956
00957
         int argc,
00958
        char *argv[],
        const char *varname,
00959
00960
        int arridx.
```

```
const char *defvalue,
00962
        char *value) {
00963
00964
        FILE *in = NULL;
00965
00966
        char dummy [LEN], fullname1 [LEN], fullname2 [LEN], line [LEN],
          msg[LEN], rvarname[LEN], rval[LEN];
00968
00969
        int contain = 0, i;
00970
00971
         /* Open file... */
        if (filename[strlen(filename) - 1] != '-')
if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");
00972
00973
00974
00975
00976
         /\star Set full variable name... \star/
00977
        if (arridx >= 0) {
          sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
00978
00980
        sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
00981
00982
00983
00984
00985
        /* Read data... */
00986
        if (in != NULL)
00987
          while (fgets(line, LEN, in))
            if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
00988
               if (strcasecmp(rvarname, fullname1) == 0 | |
00989
                   strcasecmp(rvarname, fullname2) == 0) {
00990
00991
                 contain = 1;
00992
                 break;
00993
        for (i = 1; i < argc - 1; i++)</pre>
00994
         if (strcasecmp(argv[i], fullname1) == 0 ||
    strcasecmp(argv[i], fullname2) == 0) {
    sprintf(rval, "%s", argv[i + 1]);
00995
00996
00997
             contain = 1;
00999
             break;
01000
01001
        /* Close file... */
01002
        if (in != NULL)
01003
01004
          fclose(in);
01005
01006
        /* Check for missing variables... */
01007
        if (!contain) {
         if (strlen(defvalue) > 0)
01008
            sprintf(rval, "%s", defvalue);
01009
01010
           else {
01011
            sprintf(msg, "Missing variable %s!\n", fullname1);
01012
             ERRMSG(msg);
01013
01014
        }
01015
        /* Write info... */
printf("%s = %s\n", fullname1, rval);
01016
01018
01019
         /* Return values... */
        if (value != NULL)
  sprintf(value, "%s", rval);
01020
01021
01022
        return atof(rval);
01023 }
01024
01026
01027 void time2jsec(
01028
        int year,
01029
        int mon.
        int day,
01031
        int hour,
01032
        int min,
01033
        int sec,
01034
        double remain,
01035
        double *jsec) {
01036
01037
        struct tm t0, t1;
01038
01039
        t0.tm\_year = 100;
01040
        t0.tm_mon = 0;
        t0.tm_mday = 1;
01041
        t0.tm\_hour = 0;
01042
01043
        t0.tm_min = 0;
        t0.tm\_sec = 0;
01044
01045
01046
        t1.tm_year = year - 1900;
01047
        t1.tm_mon = mon - 1;
```

```
t1.tm_mday = day;
           t1.tm_hour = hour;
01049
01050
          t1.tm_min = min;
01051
          t1.tm_sec = sec;
01052
           *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01053
01055
01057
01058 void timer(
01059
         const char *name.
01060
           int id,
01061
          int mode) {
01062
01063
          static double starttime[NTIMER], runtime[NTIMER];
01064
01065
          /* Check id... */
if (id < 0 || id >= NTIMER)
01066
             ERRMSG("Too many timers!");
01068
01069
           /* Start timer...
01070
          if (mode == 1) {
            if (starttime[id] <= 0)</pre>
01071
01072
                starttime[id] = omp_get_wtime();
01073
01074
                ERRMSG("Timer already started!");
01075
01076
01077
           /* Stop timer... */
01078
           else if (mode == 2) {
            if (starttime[id] > 0) {
01080
               runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
01081
                starttime[id] = -1;
01082
               ERRMSG("Timer not started!");
01083
01084
          }
01086
          /* Print timer...
01087
          else if (mode == 3)
01088
             printf("%s = %g sn", name, runtime[id]);
01089 }
01090
01093 double tropopause(
01094 double t,
01095
          double lat) {
01096
01097
          static double dovs[12]
          = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01099
01100
          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, -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,
01101
01102
01103
01105
01106
              45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
01107
             75, 77.5, 80, 82.5, 85, 87.5, 90
01108
01109
01110
          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,
01111
01112
01113
                     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, 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,
01114
01115
01116
01117
           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,
01118
01119
            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,
01120
01121
            98.88, 98.52, 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.
01122
01124
           284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2, 287.5, 286.2, 285.8}, {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
01125
01126
01127
            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,
01128
            100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2, 99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
01130
01131
            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, 306.6, 306.2, 306},
01132
01133
01134
```

```
{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,
01136
01137
            102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
01138
            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,
01139
01140
            263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4
01141
            315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
01142
01143
           {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,
01144
01145
            101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
01146
            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,
01147
01148
01149
            273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01150
            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,
01151
01152
            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,
01155
01156
            127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
01157
            251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
           308.5, 312.2, 313.1, 313.3}, {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
01158
01159
            187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
01160
            235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
01161
01162
            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,
01163
01164
01165
            275.4, 281.1,
                               286.3, 292, 296.3, 298.2, 298.8},
01166
           {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01167
01168
            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,
01169
01170
            112.2, 112.5, 112.6, 113, 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,
01171
            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},
01174
01175
           {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,
01176
            243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5, 114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
01177
01178
            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,
01179
01180
01181
            203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5
           276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1}, {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5, 215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2, 237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
01182
01183
01184
            111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01186
01187
            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,
01188
01189
            279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
01190
            305.1},
           {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01192
01193
            253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
01194
            223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
            108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5, 102.5, 102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
01195
01196
01197
            109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8,
            241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6, 286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6}
01198
01199
01200
           {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
01201
            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,
01202
            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,
01203
01205
01206
            280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
01207
            281.7, 281.1, 281.2}
01208
01209
01210
          double doy, p0, p1, pt;
01211
01212
          int imon, ilat;
01213
           /* Get day of year... */
01214
          doy = fmod(t / 86400., 365.25);
01215
          while (doy < 0)
             doy += 365.25;
01217
01218
           /* Get indices... */
01219
          imon = locate(doys, 12, doy);
ilat = locate(lats, 73, lat);
01220
```

```
01222
01223
        /* Get tropopause pressure... */
01224
       p0 = LIN(lats[ilat], tps[imon][ilat],
01225
                 lats[ilat + 1], tps[imon][ilat + 1], lat);
01226
       01227
       pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01228
01229
01230
       /* Return tropopause pressure... */
01231
       return pt;
01232 }
01233
01235
01236 void write_atm(
01237
       const char *filename,
       ctl_t * ctl,
atm_t * atm,
01238
01239
01240
       double t) {
01241
01242
       FILE *in, *out;
01243
01244
       char line[LEN];
01245
01246
       double r, t0, t1;
01247
01248
       int ip, iq, year, mon, day, hour, min, sec;
01249
01250
       /* Set time interval for output... */
01251
       t0 = t - 0.5 * ctl->dt_mod;

t1 = t + 0.5 * ctl->dt_mod;
01252
01253
01254
       /\star Check if gnuplot output is requested... \star/
01255
        if (ctl->atm_gpfile[0] != '-') {
01256
         /* Write info... */
01257
01258
         printf("Plot atmospheric data: %s.png\n", filename);
01259
         /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
01260
01261
           ERRMSG("Cannot create pipe to gnuplot!");
01262
01263
         /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
01264
01265
01266
01267
          /* Set time string... */
          jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01268
01269
01270
                  year, mon, day, hour, min);
01271
          /* Dump gnuplot file to pipe... */
         if (!(in = fopen(ctl->atm_gpfile, "r")))
01273
01274
           ERRMSG("Cannot open file!");
         while (fgets(line, LEN, in))
fprintf(out, "%s", line);
01275
01276
01277
         fclose(in);
01278
01279
01280
       else {
01281
01282
         /* Write info... */
01283
         printf("Write atmospheric data: %s\n", filename);
01284
01285
          /* Create file... */
01286
         if (!(out = fopen(filename, "w")))
           ERRMSG("Cannot create file!");
01287
01288
01289
01290
       /* Write header... */
01291
       fprintf(out,
01292
                "# $1 = time [s] \n"
                "# $2 = altitude [km] \n"
01293
               "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01294
       for (iq = 0; iq < ctl->nq; iq++)
fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
01295
01296
                  ctl->qnt_unit[iq]);
01297
01298
       fprintf(out, "\n");
01299
        /* Write data... */
01300
       for (ip = 0; ip < atm->np; ip++) {
01301
01302
01303
          /* Check time... */
01304
         if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
01305
            continue;
01306
         01307
01308
```

```
01309
                  atm->lon[ip], atm->lat[ip]);
          for (iq = 0; iq < ctl->nq; iq++) {
    fprintf(out, " ");
01310
01311
            fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01312
01313
01314
          fprintf(out, "\n");
01315
01316
01317
        /* Close file... */
01318
        fclose(out);
01319 }
01320
01322
01323 void write_csi(
01324
       const char *filename,
        ctl_t * ctl,
atm_t * atm,
01325
01326
01327
        double t) {
01328
01329
        static FILE *in, *out;
01330
01331
        static char line[LEN];
01332
01333
        static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01334
         rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01335
01336
        static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01337
01338
        /* Init... */
01339
        if (!init) {
01340
          init = 1;
01341
01342
          /\star Check quantity index for mass... \star/
          if (ctl->qnt_m < 0)
    ERRMSG("Need quantity mass to analyze CSI!");</pre>
01343
01344
01345
01346
          /* Open observation data file... */
          printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
if (!(in = fopen(ctl->csi_obsfile, "r")))
01347
01348
            ERRMSG("Cannot open file!");
01349
01350
          /* Create new file... */
printf("Write CSI data: %s\n", filename);
01351
01352
01353
          if (!(out = fopen(filename, "w")))
01354
            ERRMSG("Cannot create file!");
01355
          /* Write header... */
01356
01357
          fprintf(out,
01358
                   "# $1 = time [s]\n"
                   "# $2 = number of hits (cx)\n"
01359
01360
                   "# $3 = number of misses (cy)\n"
01361
                   "# $4 = number of false alarms (cz)\n"
                   01362
01363
                   "# $7 = bias (forecasts/observations) [%%]\n"
01364
01365
                   "# $8 = probability of detection (POD) [%%]\n"
01366
                   "# $9 = false alarm rate (FAR) [%%]\n"
01367
                   "# $10 = critical success index (CSI) [%%]\n\n");
01368
01369
        /* Set time interval... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01370
01371
01372
01373
01374
        /* Initialize grid cells... */
01375
        for (ix = 0; ix < ctl->csi_nx; ix++)
         for (iy = 0; iy < ctl->csi_ny; iy++)
for (iz = 0; iz < ctl->csi_nz; iz++)
01376
01377
01378
              modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01379
01380
        /* Read data... */
01381
        while (fgets(line, LEN, in)) {
01382
          /* Read data... */
if (sscanf(line, "%lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01383
01384
01385
              5)
01386
            continue;
01387
          /* Check time... */
01388
01389
          if (rt < t0)</pre>
01390
            continue;
01391
          if (rt > t1)
01392
            break;
01393
01394
          /* Calculate indices... */
          ix = (int) ((rlon - ctl->csi_lon0)
01395
```

```
01396
                         / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
           iy = (int) ((rlat - ctl->csi_lat0)
01397
01398
                         / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
           iz = (int) ((rz - ctl - csi_z0))
01399
01400
                         / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01401
01402
           /* Check indices... */
01403
           if (ix < 0 || ix >= ctl->csi_nx ||
01404
               iy < 0 \mid \mid iy >= ctl->csi_ny \mid \mid iz < 0 \mid \mid iz >= ctl->csi_nz)
             continue;
01405
01406
          /* Get mean observation index... */
01407
01408
           obsmean[ix][iy][iz] += robs;
01409
           obscount[ix][iy][iz]++;
01410
01411
         /* Analyze model data... */
for (ip = 0; ip < atm->np; ip++) {
01412
01413
01414
          /* Check time... */
if (atm->time[ip] < t0 || atm->time[ip] > t1)
01415
01416
01417
             continue;
01418
01419
           /* Get indices... */
          ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01420
                         / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01421
01422
           iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
01423
                         / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
           01424
01425
01426
01427
           /* Check indices... */
01428
           if (ix < 0 || ix >= ctl->csi_nx ||
01429
               iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01430
             continue;
01431
          /* Get total mass in grid cell... */
modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01432
01433
01434
01435
01436
         /* Analyze all grid cells... */
         for (ix = 0; ix < ctl->csi_nx; ix++)
  for (iy = 0; iy < ctl->csi_ny; iy++)
    for (iz = 0; iz < ctl->csi_nz; iz++) {
01437
01438
01439
01440
01441
                /* Calculate mean observation index... */
01442
               if (obscount[ix][iy][iz] > 0)
01443
                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01444
01445
               /* Calculate column density... */
01446
               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;
01447
01448
                 lat = ctl->csi_lat0 + dlat * (iy + 0.5);
area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
* cos(lat * M_PI / 180.);
01449
01450
01451
                  modmean[ix][iy][iz] /= (1e6 * area);
01453
01454
               /* Calculate CSI... */
01455
               if (obscount[ix][iy][iz] > 0) {
01456
                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01457
01458
                      modmean[ix][iy][iz] >= ctl->csi_modmin)
01459
01460
                  else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01461
                            modmean[ix][iy][iz] < ctl->csi_modmin)
01462
                    су++;
                  else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
01463
01464
                            modmean[ix][iy][iz] >= ctl->csi_modmin)
01465
                    cz++;
01466
01467
01468
        /* Write output... */
01469
         if (fmod(t, ctl->csi_dt_out) == 0) {
01470
01471
01472
01473
           fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
                    (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN, (cx + cz > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
01474
01475
01476
01478
                    (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01479
01480
           /\star Set counters to zero... \star/
01481
          cx = cy = cz = 0;
01482
```

```
01483
01484
        /* Close file... */
01485
        if (t == ctl->t_stop)
01486
          fclose(out);
01487 }
01488
01490
01491 void write_ens(
01492
        const char *filename,
        ctl_t * ctl,
atm_t * atm,
01493
01494
01495
        double t) {
01496
01497
        static FILE *out;
01498
        static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
01499
          t0, t1, x[NENS][3], xm[3];
01500
01501
01502
        static int init, ip, iq;
01503
01504
        static size_t i, n;
01505
01506
        /* Init... */
if (!init) {
01507
01508
          init = 1;
01509
01510
           /* Check quantities... */
          if (ctl->qnt_ens < 0)
   ERRMSG("Missing ensemble IDs!");</pre>
01511
01512
01513
01514
           /* Create new file... */
          printf("Write ensemble data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01515
01516
            ERRMSG("Cannot create file!");
01517
01518
           /* Write header... */
01519
01520
           fprintf(out,
01521
                    "# $1 = time [s] \n"
                    "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01522
01523
          01524
01525
01526
           for (iq = 0; iq < ctl->nq; iq+)
fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
01527
01528
01529
                      ctl->qnt_name[iq], ctl->qnt_unit[iq]);
          fprintf(out, "# \$%d = number of members\n\n", 5 + 2 * ctl->nq);
01530
01531
01532
01533
         /* Set time interval... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01534
01535
01536
        /* Init... */
01537
01538
        ens = GSL_NAN;
        n = 0;
01540
        /* Loop over air parcels... */
for (ip = 0; ip < atm->np; ip++) {
01541
01542
01543
01544
           /* Check time... */
01545
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
01546
            continue;
01547
01548
           /\star Check ensemble id... \star/
          if (atm->q[ctl->qnt_ens][ip] != ens) {
01549
01550
01551
             /* Write results... */
             if (n > 0) {
01553
01554
               /* Get mean position... */
01555
               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>
01556
01557
01558
01559
                 xm[2] += x[i][2] / (double) n;
01560
               cart2geo(xm, &dummy, &lon, &lat);
fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
01561
01562
01563
                        lat);
01564
01565
                /* Get quantity statistics... */
               for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01566
01567
01568
                 fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01569
```

```
for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
                 fprintf(out, " ");
fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01571
01572
01573
01574
                fprintf(out, " lu\n", n);
01575
01576
01577
             /* Init new ensemble... */
01578
             ens = atm->q[ctl->qnt_ens][ip];
01579
             n = 0;
           }
01580
01581
01582
           /* Save data... */
01583
           p[n] = atm -> p[ip];
01584
           geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
           for (iq = 0; iq < ctl->nq; iq++)
  q[iq][n] = atm->q[iq][ip];
if ((++n) >= NENS)
01585
01586
01587
             ERRMSG("Too many data points!");
01588
01589
01590
01591
         /* Write results... */
01592
        if (n > 0) {
01593
01594
           /* Get mean position... */
01595
           xm[0] = xm[1] = xm[2] = 0;
01596
           for (i = 0; i < n; i++) {</pre>
            xm[0] += x[i][0] / (double) n;

xm[1] += x[i][1] / (double) n;
01597
01598
             xm[2] += x[i][2] / (double) n;
01599
01600
           cart2geo(xm, &dummy, &lon, &lat);
fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
01601
01602
01603
01604
            /* Get quantity statistics... */
           for (iq = 0; iq < ctl->nq; iq++) {
   fprintf(out, " ");
   fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01605
01606
01607
01608
           for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01609
01610
             fprintf(out,
             fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01611
01612
01613
           fprintf(out, " %lu\n", n);
01614
01615
01616
         /* Close file... */
         if (t == ctl->t_stop)
01617
           fclose(out);
01618
01619 }
01620
01622
01623 void write_grid(
01624
        const char *filename,
        ctl_t * ctl,
met_t * met0,
01625
01626
01627
01628
        atm_t * atm,
01629
        double t) {
01630
01631
        FILE *in, *out;
01632
01633
        char line[LEN];
01634
01635
        static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01636
           area, rho_air, press, temp, cd, mmr, t0, t1, r;
01637
01638
        static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01639
01640
         /* Check dimensions... */
01641
         if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
01642
          ERRMSG("Grid dimensions too large!");
01643
        /* Check quantity index for mass... */
if (ctl->qnt_m < 0)</pre>
01644
01645
01646
           ERRMSG("Need quantity mass to write grid data!");
01647
        /* Set time interval for output... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01648
01649
01650
01651
01652
         /* Set grid box size...
01653
         dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
        dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
01654
01655
01656
```

```
/* Initialize grid... */
01658
         for (ix = 0; ix < ctl->grid_nx; ix++)
01659
          for (iy = 0; iy < ctl->grid_ny; iy++)
             for (iz = 0; iz < ctl->grid_nz; iz++)
01660
                grid_m[ix][iy][iz] = 0;
01661
01662
01663
          /* Average data... */
01664
         for (ip = 0; ip < atm->np; ip++)
01665
           if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {</pre>
01666
01667
              /* Get index... */
              ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
01668
01669
01670
              iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01671
              /* Check indices... */
01672
              if (ix < 0 || ix >= ctl->grid_nx ||
    iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01673
01674
                 continue;
01675
01676
              /* Add mass... */
01677
01678
              grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01679
01680
01681
         /* Check if gnuplot output is requested... */
         if (ctl->grid_gpfile[0] != '-') {
01682
01683
01684
            /* Write info... */
            printf("Plot grid data: %s.png\n", filename);
01685
01686
01687
            /* Create gnuplot pipe... */
           if (!(out = popen("gnuplot", "w")))
ERRMSG("Cannot create pipe to gnuplot!");
01688
01689
01690
           /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
01691
01692
01693
01694
            /* Set time string... */
            jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01695
01696
01697
                     year, mon, day, hour, min);
01698
            /* Dump gnuplot file to pipe... */
01699
           if (!(in = fopen(ctl->grid_gpfile, "r")))
01700
01701
             ERRMSG("Cannot open file!");
            while (fgets(line, LEN, in))
fprintf(out, "%s", line);
01702
01703
01704
           fclose(in);
         }
01705
01706
         else {
01708
01709
            /* Write info... */
01710
           printf("Write grid data: %s\n", filename);
01711
01712
            /* Create file... *
01713
           if (!(out = fopen(filename, "w")))
01714
              ERRMSG("Cannot create file!");
01715
01716
         /* Write header... */
01717
01718
         fprintf(out,
                   "# $1 = time [s] \n"
01720
                   "# $2 = altitude [km] \n"
01721
                   "# $3 = longitude [deg] \n"
                   "# $4 = latitude [deg] \n"
01722
                   "# $5 = surface area [km^2]\n"
01723
                   "# $6 = layer width [km] \n"
01724
                   "# $7 = temperature [K]\n"
01725
                   "# $8 = \text{column density } [kg/m^2] \n"
01726
01727
                   "# $9 = mass mixing ratio [1]\n\n");
01728
         /* Write data... */
for (ix = 0; ix < ctl->grid_nx; ix++) {
   if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01729
01730
01731
01732
             fprintf(out, "\n");
01733
            for (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++)
01734
01735
01736
01737
                if (!ctl->grid_sparse
                      || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01739
                   /* Set coordinates... */
01740
                   z = ctl->grid_z0 + dz * (iz + 0.5);
lon = ctl->grid_lon0 + dlon * (ix + 0.5);
lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01741
01742
01743
```

```
01744
01745
                  /\star Get pressure and temperature... \star/
01746
                  press = P(z);
01747
                  intpol_met_time(met0, met1, t, press, lon, lat,
01748
                                   NULL, &temp, NULL, NULL, NULL, NULL, NULL);
01749
01750
                 /* Calculate surface area... */
                 area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
 * cos(lat * M_PI / 180.);
01751
01752
01753
                 /* Calculate column density... */
cd = grid_m[ix][iy][iz] / (1e6 * area);
01754
01755
01756
                 /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01757
01758
01759
01760
01761
                  /* Write output...
                 01762
01763
01764
01765
          }
01766 }
01767
01768
        /* Close file... */
01769
        fclose(out);
01770 }
01771
01773
01774 void write_prof(
01775
        const char *filename,
01776
        ctl_t * ctl,
01777
        met_t * met0,
        met_t * met1,
atm_t * atm,
01778
01779
01780
        double t) {
01781
01782
        static FILE *in, *out;
01783
01784
        static char line[LEN];
01785
01786
        static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
01787
          rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z,
01788
          press, temp, rho_air, mmr, h2o, o3;
01789
01790
        static int init, obscount[GX][GY], ip, ix, iy, iz;
01791
01792
        /* Init... */
01793
        if (!init) {
01794
          init = 1;
01795
01796
           /\star Check quantity index for mass... \star/
          if (ctl->qnt_m < 0)
   ERRMSG("Need quantity mass!");</pre>
01797
01798
01799
01800
           /* Check dimensions... */
01801
          if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
             ERRMSG("Grid dimensions too large!");
01802
01803
           /* Open observation data file... */
01804
           printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01805
01806
           if (!(in = fopen(ctl->prof_obsfile, "r")))
            ERRMSG("Cannot open file!");
01807
01808
01809
           /* Create new file... */
          printf("Write profile data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01810
01811
             ERRMSG("Cannot create file!");
01812
01813
01814
           /* Write header... */
01815
           fprintf(out,
                    "# $1 = time [s]\n"
"# $2 = altitude [km]\n"
01816
                    "# $1
01817
                    "# $3 = longitude [deg] \n"
01818
01819
                    "# $4 = latitude [deg]\n"
01820
                    "# $5
                           = pressure [hPa]\n"
01821
                    "# $6 = temperature [K]\n"
                    "# $7 = mass mixing ratio [1]\n"
01822
                    "# $8 = H2O volume mixing ratio [1]\n"
"# $9 = O3 volume mixing ratio [1]\n"
01823
01824
                    "# $10 = mean BT index [K]\n");
01825
01826
01827
           /* 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;
01828
01829
01830
```

```
01831
         }
01832
01833
         /* Set time interval... */
         t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01834
01835
01836
01837
         /* Initialize... */
01838
         for (ix = 0; ix < ctl->prof_nx; ix++)
01839
          for (iy = 0; iy < ctl->prof_ny; iy++) {
01840
             obsmean[ix][iy] = 0;
01841
              obscount[ix][iy] = 0;
01842
             tmean[ix][iy] = 0;
for (iz = 0; iz < ctl->prof_nz; iz++)
01843
01844
               mass[ix][iy][iz] = 0;
01845
01846
        /* Read data... */
while (fgets(line, LEN, in)) {
01847
01848
01849
01850
           /* Read data... */
           if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01851
01852
              continue;
01853
           /* Check time... */
01854
01855
           <u>if</u> (rt < t0)
01856
             continue;
01857
           if (rt > t1)
01858
             break;
01859
           /* Calculate indices... */
ix = (int) ((rlon - ctl->prof_lon0) / dlon);
01860
01861
01862
           iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01863
01864
           /\star Check indices... \star/
           if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01865
01866
             continue;
01867
01868
           /* Get mean observation index... */
           obsmean[ix][iy] += robs;
tmean[ix][iy] += rt;
01869
01870
01871
           obscount[ix][iy]++;
01872
01873
01874
         /* Analyze model data... */
01875
         for (ip = 0; ip < atm->np; ip++) {
01876
01877
           /* Check time... */
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
01878
01879
             continue:
01880
01881
           /* 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);
01882
01883
01884
01885
01886
           /* Check indices... */
           if (ix < 0 || ix >= ctl->prof_nx ||
01888
               iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01889
01890
           /* Get total mass in grid cell... */
mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01891
01892
01893
01894
01895
         /* Extract profiles... */
01896
         for (ix = 0; ix < ctl->prof_nx; ix++)
          for (iy = 0; iy < ctl->prof_ny; iy++)
01897
             if (obscount[ix][iy] > 0) {
01898
01899
                /* Write output... */
01901
                fprintf(out, "\n");
01902
01903
                /* Loop over altitudes... */
01904
                for (iz = 0; iz < ctl->prof_nz; iz++) {
01905
01906
                  /* Set coordinates... */
01907
                  z = ctl - prof_z0 + dz * (iz + 0.5);
                  lon = ctl->prof_lon0 + dlon * (ix + 0.5);
lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01908
01909
01910
01911
                  /* Get meteorological data... */
01912
                  press = P(z);
                  01913
01914
01915
                  /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
01916
01917
```

```
area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
* cos(lat * M_PI / 180.);
01919
01920
               mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
01921
               01922
01923
01924
01925
                       z, lon, lat, press, temp, mmr, h2o, o3,
01926
                       obsmean[ix][iy] / obscount[ix][iy]);
01927
           }
01928
01929
01930
       /* Close file... */
       if (t == ctl->t_stop)
01931
01932
         fclose(out);
01933 }
01934
01936
01937 void write_station(
01938 const char *filename,
       ctl_t * ctl,
atm_t * atm,
01939
01940
01941
       double t) {
01942
01943
       static FILE *out;
01944
01945
       static double rmax2, t0, t1, x0[3], x1[3];
01946
01947
       static int init, ip, iq;
01948
01949
       /* Init... */
01950
       if (!init) {
01951
         init = 1;
01952
         /* Write info... */
01953
         printf("Write station data: %s\n", filename);
01954
01955
01956
          /* Create new file...
         if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");
01957
01958
01959
          /* Write header... */
01960
01961
         fprintf(out,
                  "# $1 = time [s]\n"
01962
01963
                  "# $2 = altitude [km] \n"
                 "# $3 = longitude [deg] \n" "# $4 = latitude [deg] \n");
01964
         01965
01966
01967
         fprintf(out, "\n");
01968
01969
01970
          /\star Set geolocation and search radius... \star/
01971
         geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
01972
         rmax2 = gsl_pow_2(ctl->stat_r);
01973
01974
01975
        /* Set time interval for output... */
       t0 = t - 0.5 * ctl->dt_mod;

t1 = t + 0.5 * ctl->dt_mod;
01976
01977
01978
01979
       /* Loop over air parcels... */
01980
       for (ip = 0; ip < atm->np; ip++) {
01981
          /\star Check time... \star/
01982
01983
         if (atm->time[ip] < t0 || atm->time[ip] > t1)
01984
           continue;
01985
         /* Check station flag... */
01986
         if (ctl->qnt_stat >= 0)
01988
          if (atm->q[ctl->qnt_stat][ip])
             continue;
01989
01990
01991
         /* Get Cartesian coordinates... */
01992
         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
01993
01994
          /* Check horizontal distance... */
01995
         if (DIST2(x0, x1) > rmax2)
01996
           continue;
01997
01998
         /* Set station flag... */
01999
         if (ctl->qnt_stat >= 0)
02000
           atm->q[ctl->qnt_stat][ip] = 1;
02001
          /* Write data... */
fprintf(out, "%.2f %g %g %g",
02002
02003
02004
                 atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
```

## 5.13 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 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 \*t, double \*u, double \*v, double \*w, 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 \*t, double \*u, double \*v, double \*w, 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\_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.

 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.

- double tropopause (double t, double lat)
- 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.

# 5.13.1 Detailed Description

MPTRAC library declarations.

Definition in file libtrac.h.

## 5.13.2 Function Documentation

```
5.13.2.1 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 }
```

5.13.2.2 double deg2dx ( double dlon, double lat )

Convert degrees to horizontal distance.

Definition at line 45 of file libtrac.c.

5.13.2.3 double deg2dy ( double dlat )

Convert degrees to horizontal distance.

Definition at line 54 of file libtrac.c.

5.13.2.4 double dp2dz ( double dp, double p )

Convert pressure to vertical distance.

Definition at line 62 of file libtrac.c.

```
00064 {
00065
00066 return -dp * H0 / p;
00067 }
```

# 5.13.2.5 double dx2deg ( double dx, double lat )

Convert horizontal distance to degrees.

Definition at line 71 of file libtrac.c.

## 5.13.2.6 double dy2deg ( double dy )

Convert horizontal distance to degrees.

Definition at line 84 of file libtrac.c.

#### 5.13.2.7 double dz2dp ( double dz, double p )

Convert vertical distance to pressure.

Definition at line 92 of file libtrac.c.

```
00094 {
00095
00096 return -dz * p / H0;
00097 }
```

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

Convert geolocation to Cartesian coordinates.

Definition at line 101 of file libtrac.c.

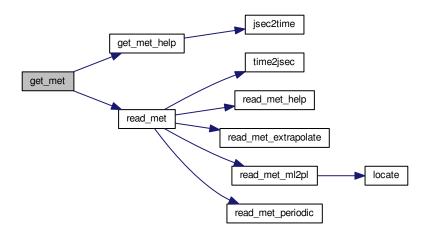
5.13.2.9 void get\_met ( ctl\_t \* ctl, char \* metbase, double t, met\_t \* met0, met\_t \* met1 )

Get meteorological data for given timestep.

Definition at line 117 of file libtrac.c.

```
00122
00123
00124
        char filename[LEN];
00125
00126
        static int init;
00127
        /* Init... */
00128
        if (!init) {
00129
00130
          init = 1;
00131
00132
          get_met_help(t, -1, metbase, ctl->dt_met, filename);
00133
          read_met(ctl, filename, met0);
00134
          get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
00135
      dt_met, filename);
00136
          read_met(ctl, filename, metl);
00137
00138
        /\star Read new data for forward trajectories... \star/
00139
00140
        if (t > met1->time && ct1->direction == 1) {
         memcpy(met0, met1, sizeof(met_t));
00141
          get_met_help(t, 1, metbase, ctl->dt_met, filename);
read_met(ctl, filename, met1);
00142
00143
00144
00145
00146
        /* Read new data for backward trajectories... */
        if (t < met0->time && ctl->direction == -1) {
00147
         memcpy(met1, met0, sizeof(met_t));
          get_met_help(t, -1, metbase, ctl->dt_met, filename);
read_met(ctl, filename, met0);
00149
00150
00151
00152 }
```

Here is the call graph for this function:



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

Get meteorological data for timestep.

Definition at line 156 of file libtrac.c.

```
{
00163
        double t6, r;
00164
00165
        int year, mon, day, hour, min, sec;
00166
00167
        /\star Round time to fixed intervals... \star/
00168
        if (direct == -1)
00169
         t6 = floor(t / dt_met) * dt_met;
00170
        else
00171
          t6 = ceil(t / dt_met) * dt_met;
00172
00173
        /* Decode time... */
00174
       jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00175
       /* Set filename... */
sprintf(filename, "%s_%d_%02d_%02d_nc", metbase, year, mon, day, hour);
00176
00177
00178 }
```

Here is the call graph for this function:



5.13.2.11 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 182 of file libtrac.c.

```
00188
00189
         double aux00, aux01, aux10, aux11;
00190
00191
00192
          /* Set variables...
         aux00 = array[ix][iy];
aux01 = array[ix][iy + 1];
aux10 = array[ix + 1][iy];
00193
00194
00195
00196
          aux11 = array[ix + 1][iy + 1];
00197
00198
         /* Interpolate horizontally... */
00199 aux00 = wy * (aux00 - aux01) + aux01;
00200 aux11 = wy * (aux10 - aux11) + aux11;
00201
          *var = wx * (aux00 - aux11) + aux11;
00202 }
```

5.13.2.12 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 206 of file libtrac.c.

```
00214
00215
00216
       double aux00, aux01, aux10, aux11;
00217
       /* Interpolate vertically... */
00218
00219
      aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
        + array[ix][iy][ip + 1];
00220
00221
       aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
      00222
00223
00224
00225
       + array[ix + 1][iy + 1][ip + 1];
00226
00227
00228
      /* Interpolate horizontally... */
      aux00 = wy * (aux00 - aux01) + aux01;

aux11 = wy * (aux10 - aux11) + aux11;
00229
00230
       *var = wx * (aux00 - aux11) + aux11;
00231
00232 }
```

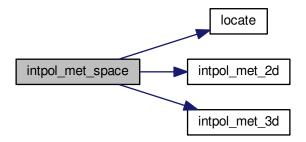
5.13.2.13 void intpol\_met\_space ( met\_t \* met, double p, double lon, double lon

Spatial interpolation of meteorological data.

Definition at line 236 of file libtrac.c.

```
00247
00248
00249
        double wp, wx, wv;
00250
00251
         int ip, ix, iy;
00252
        /* Check longitude... */
if (met->lon[met->nx - 1] > 180 && lon < 0)
00253
00254
00255
          lon += 360;
00256
00257
         /* Get indices... */
00258
         ip = locate(met->p, met->np, p);
00259
         ix = locate(met->lon, met->nx, lon);
00260
         iy = locate(met->lat, met->ny, lat);
00261
00262
         /* Get weights... */
        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]);
00263
00264
00265
00266
00267
         /* Interpolate... */
         if (ps != NULL)
00268
           intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
00269
00270
         if (t != NULL)
00271
           intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
00272
         if (u != NULL)
00273
           intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
00274
         if (v != NULL)
00275
           intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
00276
         if (w != NULL)
           intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
00277
00278
         if (h2o != NULL)
00279
        intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
if (o3 != NULL)
00280
00281
           intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00282 }
```

Here is the call graph for this function:



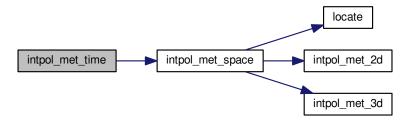
5.13.2.14 void intpol\_met\_time ( met\_t \* met0, met\_t \* met1, double ts, double p, double lon, double lat, double \* ps, double \* t, double \* u, double \* v, double \* w, double \* h2o, double \* o3 )

Temporal interpolation of meteorological data.

Definition at line 286 of file libtrac.c.

```
00299
00300
00301
       double h2o0, h2o1, o30, o31, ps0, ps1, t0, t1, u0, u1, v0, v1, w0, w1, wt;
00302
00303
        /* Spatial interpolation... */
00304
       00305
00306
                        t == NULL ? NULL : &t0,
00307
                        u == NULL ? NULL : &u0,
00308
                        v == NULL ? NULL : &v0,
00309
                        w == NULL ? NULL : &w0,
                        h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00310
       00311
00312
00313
                        t == NULL ? NULL : &t1,
00314
                        u == NULL ? NULL : &u1,
00315
                        v == NULL ? NULL : &v1,
00316
                        w == NULL ? NULL : &w1,
00317
                        h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00318
       /* Get weighting factor... */ wt = (met1->time - ts) / (met1->time - met0->time);
00319
00320
00321
00322
       /* Interpolate... */
00323
       if (ps != NULL)
         *ps = wt * (ps0 - ps1) + ps1;
00324
       if (t != NULL)
00325
       *t = wt * (t0 - t1) + t1;
if (u != NULL)
00326
00327
         *u = wt * (u0 - u1) + u1;
00328
       if (v != NULL)
00329
         *v = wt * (v0 - v1) + v1;
00330
00331
       if (w != NULL)
00332
         \star w = wt \star (w0 - w1) + w1;
       if (h2o != NULL)
00333
         *h2o = wt * (h2o0 - h2o1) + h2o1;
00334
       if (o3 != NULL)
00335
00336
         *o3 = wt * (o30 - o31) + o31;
00337 }
```

Here is the call graph for this function:



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

Convert seconds to date.

Definition at line 341 of file libtrac.c.

```
00349
00350
        struct tm t0, *t1;
00352
00353
        time_t jsec0;
00354
00355
        t0.tm_year = 100;
00356
        t0.tm_mon = 0;
00357
        t0.tm_mday = 1;
00358
        t0.tm\_hour = 0;
00359
        t0.tm_min = 0;
        t0.tm_sec = 0;
00360
00361
00362
        jsec0 = (time_t) jsec + timegm(&t0);
00363
        t1 = gmtime(&jsec0);
00364
00365
        *year = t1->tm_year + 1900;
        *mon = t1->tm_mon + 1;
*day = t1->tm_mday;
00366
00367
        *hour = t1->tm_hour;
00368
        *min = t1->tm_min;
00369
00370
        *sec = t1->tm_sec;
00371
        *remain = jsec - floor(jsec);
00372 }
```

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

Find array index.

Definition at line 376 of file libtrac.c.

```
00379
00380
00381
          int i, ilo, ihi;
00382
00383
          ilo = 0;
00384
         ihi = n - 1;
00385
          i = (ihi + ilo) >> 1;
00386
         if (xx[i] < xx[i + 1])
  while (ihi > ilo + 1) {
   i = (ihi + ilo) >> 1;
00387
00388
00389
00390
               if (xx[i] > x)
```

```
ihi = i;
00392
00393
               ilo = i;
00394
        } else
          while (ihi > ilo + 1) {
  i = (ihi + ilo) >> 1;
00395
00396
             if (xx[i] <= x)
00398
               ihi = i;
00399
             else
00400
               ilo = i;
         }
00401
00402
00403
        return ilo;
```

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

Read atmospheric data.

Definition at line 408 of file libtrac.c.

```
00411
00412
          FILE *in:
00413
00414
          char line[LEN], *tok;
00416
00417
00418
          /* Init... */
atm->np = 0;
00419
00420
00421
00422
          /* Write info... */
00423
          printf("Read atmospheric data: %s\n", filename);
00424
          /* Open file... */
if (!(in = fopen(filename, "r")))
00425
00426
             ERRMSG("Cannot open file!");
00428
00429
          /* Read line... */
00430
          while (fgets(line, LEN, in)) {
00431
             /* 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]);
00432
00433
00435
00436
00437
00438
00439
00440
             /* Convert altitude to pressure... */
00441
             atm->p[atm->np] = P(atm->p[atm->np]);
00442
             /* Increment data point counter... */
if ((++atm->np) > NP)
00443
00444
                ERRMSG("Too many data points!");
00445
00447
00448
          /* Close file... */
00449
          fclose(in);
00450
00451
           /* Check number of points... */
           if (atm->np < 1)</pre>
00452
00453
             ERRMSG("Can not read any data!");
00454 }
```

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

Read control parameters.

Definition at line 458 of file libtrac.c.

```
00462
                      {
00463
00464
        int ip, iq;
00465
        /* Write info... */
00466
        printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
00467
               "(executable: %s | compiled: %s, %s)\n\n",
00469
               argv[0], __DATE__, __TIME__);
00470
00471
        /* Initialize quantity indices... */
        ctl->qnt_ens = -1;
00472
        ctl->qnt_m = -1;
00473
        ctl->qnt_r = -1;
00474
00475
        ctl->qnt_rho = -1;
00476
        ctl->qnt_ps = -1;
        ctl->qnt_p = -1;
00477
        ctl \rightarrow qnt_t = -1;
00478
        ctl \rightarrow qnt_u = -1;
00479
00480
        ctl->qnt_v = -1;
        ctl->qnt_w = -1;
00481
00482
        ctl->qnt_h2o = -1;
00483
        ctl->qnt_o3 = -1;
00484
        ctl->qnt\_theta = -1;
        ctl->qnt\_pv = -1;
00485
00486
        ctl->qnt\_tice = -1;
        ctl \rightarrow qnt_tsts = -1;
00487
00488
        ctl->qnt\_tnat = -1;
00489
        ctl->qnt\_stat = -1;
00490
        /* Read quantities... */
00491
00492
        ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
00493
        for (iq = 0; iq < ctl->nq; iq++) {
00494
          /\star Read quantity name and format... \star/
00495
          00496
00497
00498
00500
          /* Try to identify quantity...
          if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
00501
00502
            ctl->qnt_ens = iq;
            sprintf(ctl->qnt_unit[iq], "-");
00503
          } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
00504
            ctl->qnt_m = iq;
00505
            sprintf(ctl->qnt_unit[iq], "kg");
00506
00507
          } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
            ctl->qnt_r = iq;
00508
00509
            sprintf(ctl->qnt_unit[iq], "m");
          } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
  ctl->qnt_rho = iq;
  sprintf(ctl->qnt_unit[iq], "kg/m^3");
00510
00511
00512
00513
          } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
            ctl->qnt_ps = iq;
00514
00515
            sprintf(ctl->qnt_unit[iq], "hPa");
00516
          } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
            ctl->qnt_p = iq;
sprintf(ctl->qnt_unit[iq], "hPa");
00517
00519
          } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
00520
            ctl->qnt_t = iq;
00521
            sprintf(ctl->qnt_unit[iq], "K");
          } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
  ctl->qnt_u = iq;
00522
00523
            sprintf(ctl->qnt_unit[iq], "m/s");
00525
          } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
            ctl->qnt_v = iq;
00526
00527
            sprintf(ctl->qnt_unit[iq], "m/s");
          } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
  ctl->qnt_w = iq;
00528
00529
            sprintf(ctl->qnt_unit[iq], "hPa/s");
00530
          } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
            ctl->qnt_h2o = iq;
00532
00533
            sprintf(ctl->qnt_unit[iq], "1");
00534
          } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
00535
            ctl->qnt_o3 = iq;
            sprintf(ctl->qnt_unit[iq], "1");
00536
          } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
00538
            ctl->qnt_theta = iq;
00539
            sprintf(ctl->qnt_unit[iq], "K");
          } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
  ctl->qnt_pv = iq;
00540
00541
            sprintf(ctl->qnt_unit[iq], "PVU");
00542
          } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
            ctl->qnt_tice = iq;
00544
00545
            sprintf(ctl->qnt_unit[iq], "K");
00546
          } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
            ctl->qnt_tsts = iq;
sprintf(ctl->qnt_unit[iq], "K");
00547
00548
```

```
} else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
              ctl->qnt_tnat = iq;
00550
00551
               sprintf(ctl->qnt_unit[iq], "K");
00552
            } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
00553
              ctl->qnt_stat = iq;
00554
              sprintf(ctl->gnt unit[ig], "-");
           } else
              scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00556
00557
00558
00559
          /* Time steps of simulation... */
00560
          ctl->direction =
          (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
if (ctl->direction != -1 && ctl->direction != 1)
00561
00562
00563
           ERRMSG("Set DIRECTION to -1 or 1!");
          ctl->t_start =
00564
         scan_ctl(filename, argc, argv, "T_START", -1, "-le100", NULL);
ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "-le100", NULL);
ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
00565
00566
00567
00568
00569
          /* Meteorological data..
          ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL); ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
00570
00571
00572
          if (ctl->met_np > EP)
00573
            ERRMSG("Too many levels!");
          for (ip = 0; ip < ctl->met_np; ip++)
00574
            ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
00575
00576
00577
          /* Isosurface parameters... */
00578
         ctl->isosurf
         = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00579
00580
00581
00582
          /* Diffusion parameters... */
00583
         ctl->turb_dx_trop
            = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00584
00585
         ctl->turb dx strat
            = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00587
         ctl->turb_dz_trop
00588
            = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00589
         ctl->turb_dz_strat
00590
            = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00591
          ctl->turb meso =
00592
            scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00593
00594
          /* Life time of particles... */
         ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL); ctl->tdec_strat =
00595
00596
            scan ctl(filename, argc, argv, "TDEC STRAT", -1, "0", NULL);
00597
00598
00599
         /* PSC analysis... */
00600
         ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
         ctl->psc_hno3 =
00601
00602
            scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
00603
         /* Output of atmospheric data... */
scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
00604
00606
          scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00607
          ctl->atm_dt_out
            scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00608
00609
          ctl->atm filter =
00610
            (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
00611
00612
          /* Output of CSI data... */
00613
         scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
       csi_basename);
00614
         ctl->csi dt out =
         scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "obs.tab",
00615
00616
00617
                    ctl->csi_obsfile);
00618
         ctl->csi obsmin =
00619
            scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00620
          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);
00621
00622
00623
00624
          ctl->csi lon0 =
00625
         scan_ctl(filename, argc, argv, "CSI_LONO", -1, "-180", NULL);
ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
00626
00627
00628
         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);
00629
00630
00631
00632
          ctl->csi nv =
00633
             (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
```

```
00634
         /\star Output of ensemble data... \star/
00635
         scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
00636
      ens_basename);
00637
00638
         /* Output of grid data... */
00639
         scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
00640
                    ctl->grid_basename);
00641
         scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
grid_gpfile);
00642 ctl->grid_dt_out =
           scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
00643
00644
         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);
00645
00646
00647
         ctl->grid nz =
00648
           (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00649
00650
         ctl->grid_lon0 =
00651
           scan_ctl(filename, argc, argv, "GRID_LONO", -1, "-180", NULL);
00652
         ctl->grid_lon1 =
00653
           scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
00654
         ctl->grid_nx =
           (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00655
         ctl->grid_lat0 =
00656
00657
           scan_ctl(filename, argc, argv, "GRID_LATO", -1, "-90", NULL);
00658
         ctl->grid_lat1
00659
           scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
         ctl->grid_ny =
00660
00661
           (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
00662
00663
         /* Output of profile data... */
00664
        scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
00665
                    ctl->prof_basename);
prof_obsfile);

00667 ct1->-
00666
        scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
         ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
00668
00669
         ctl->prof_nz =
00670
           (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
         ctl->prof_lon0 =
00671
           scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
00672
00673
         ct.1->prof lon1
00674
           scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
00675
         ctl->prof_nx =
00676
            (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00677
         ctl->prof_lat0 =
           scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
00678
00679
         ctl->prof lat1 =
           scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
00680
00681
         ctl->prof_ny
00682
           (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00683
00684
         /* Output of station data... */
         scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
00685
00686
                   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);
00688
00689
00690 }
```

Here is the call graph for this function:



5.13.2.19 void read\_met ( ctl\_t \* ctl, char \* filename, met\_t \* met )

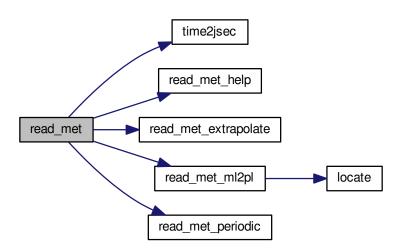
Read meteorological data file.

Definition at line 694 of file libtrac.c.

```
00697
00698
00699
         char tstr[10];
00700
00701
         static float help[EX * EY];
00702
00703
         int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00704
00705
         size_t np, nx, ny;
00706
00707
         /* Write info... */
00708
         printf("Read meteorological data: %s\n", filename);
00709
00710
          /* Get time from filename... */
00711
          sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
         year = atoi(tstr);
sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00712
00713
00714
         mon = atoi(tstr);
         sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00716
         day = atoi(tstr);
00717
          sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00718
         hour = atoi(tstr);
00719
         time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00720
00721
           * Open netCDF file... */
00722
         NC(nc_open(filename, NC_NOWRITE, &ncid));
00723
         /* Get dimensions... */
NC(nc_inq_dimid(ncid, "lon", &dimid));
          /* Get dimensions...
00724
00725
00726
         NC(nc_inq_dimlen(ncid, dimid, &nx));
00727
         if (nx > EX)
00728
            ERRMSG("Too many longitudes!");
00729
00730
         NC(nc_inq_dimid(ncid, "lat", &dimid));
00731
         NC(nc_inq_dimlen(ncid, dimid, &ny));
00732
             (nv > EY)
00733
            ERRMSG("Too many latitudes!");
00734
00735
         NC(nc_inq_dimid(ncid, "lev", &dimid));
00736
         NC(nc_inq_dimlen(ncid, dimid, &np));
00737
          if (np > EP)
00738
           ERRMSG("Too many levels!");
00739
00740
         /* Store dimensions... */
00741
         met->np = (int) np;
00742
         met->nx = (int) nx;
00743
         met->ny = (int) ny;
00744
00745
          /* Get horizontal grid... */
         NC(nc_inq_varid(ncid, "lon", &varid));
00746
         NC(nc_get_var_double(ncid, varid, met->lon));
NC(nc_inq_varid(ncid, "lat", &varid));
00747
00748
00749
         NC(nc_get_var_double(ncid, varid, met->lat));
00750
         /* 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", "W", met, met->w, 0.01f);
read_met_help(ncid, "q", "Q", met, met->help(ncid, "q", "Q", met, met->help(ncid, "o3", "o3", met, met->o3, 0.602f);
00751
00752
00753
00754
00755
00756
00757
00758
00759
          /* Meteo data on pressure levels... */
00760
         if (ctl->met_np <= 0) {</pre>
00761
            /* Read pressure levels from file... */
NC(nc_inq_varid(ncid, "lev", &varid));
00762
00763
            NC(nc_get_var_double(ncid, varid, met->p));
for (ip = 0; ip < met->np; ip++)
  met->p[ip] /= 100.;
00764
00765
00766
00767
00768
            /\star Extrapolate data for lower boundary... \star/
00769
            read_met_extrapolate(met);
00770
00771
00772
          /* Meteo data on model levels... */
00773
00774
            /* Read pressure data from file... */
read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00775
00776
00777
00778
            /\star Interpolate from model levels to pressure levels... \star/
00779
            read_met_ml2pl(ctl, met, met->t);
00780
            read_met_ml2pl(ctl, met, met->u);
00781
            read_met_ml2pl(ctl, met, met->v);
00782
            read_met_ml2pl(ctl, met, met->w);
00783
            read_met_ml2pl(ctl, met, met->h2o);
```

```
00784
             read_met_ml2pl(ctl, met, met->o3);
00785
00786
              /\star Set pressure levels... \star/
             met->np = ctl->met_np;
for (ip = 0; ip < met->np; ip++)
  met->p[ip] = ctl->met_p[ip];
00787
00788
00789
00790
00791
00792
           /\star Check ordering of pressure levels... \star/
           for (ip = 1; ip < met->np; ip++)
  if (met->p[ip - 1] < met->p[ip])
    ERRMSG("Pressure levels must be descending!");
00793
00794
00795
00796
           /* Read surface pressure... */
if (nc_ing_varid(ncid, "PS", &varid) == NC_NOERR) {
00797
00798
             NC(nc_get_var_float(ncid, varid, help));
00799
          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_ing_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00800
00801
00802
00803
00804
             NC(nc_get_var_float(ncid, varid, help));
              for (iy = 0; iy < met->ny; iy++)
  for (ix = 0; ix < met->nx; ix++)
00805
00806
                  met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00807
80800
           } else
00809
             for (ix = 0; ix < met->nx; ix++)
00810
                for (iy = 0; iy < met\rightarrowny; iy++)
00811
                   met->ps[ix][iy] = met->p[0];
00812
00813
           /\star Create periodic boundary conditions... \star/
00814
          read_met_periodic(met);
00815
00816
           /* Close file... */
00817
           NC(nc_close(ncid));
00818 }
```

Here is the call graph for this function:



5.13.2.20 void read\_met\_extrapolate ( met\_t \* met )

Extrapolate meteorological data at lower boundary.

Definition at line 822 of file libtrac.c.

```
00823
00824
00825
          int ip, ip0, ix, iy;
00826
00827
          /* Loop over columns... */
00828
          for (ix = 0; ix < met->nx; ix++)
            for (iy = 0; iy < met->ny; iy++) {
00830
00831
               /\star Find lowest valid data point... \star/
               for (ip0 = met->np - 1; ip0 >= 0; ip0--)
   if (!gsl_finite(met->t[ix][iy][ip0])
00832
00833
00834
                      | | !gsl_finite(met->u[ix][iy][ip0])
                      || !gsl_finite(met->v[ix][iy][ip0])
00835
00836
                      || !gsl_finite(met->w[ix][iy][ip0]))
00837
00838
00839
               /* Extrapolate... */
              /* Extrapolate... */
for (ip = ip0; ip >= 0; ip--) {
    met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
00840
00841
                 met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
00842
00843
                 met \rightarrow v[ix][iy][ip] = met \rightarrow v[ix][iy][ip + 1];
00844
                 met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
                met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00845
00846
00847
               }
00848
00849 }
```

5.13.2.21 void read\_met\_help ( int ncid, char \* varname, char \* varname2, met\_t \* met, float dest[EX][EY][EP], float scl )

Read and convert variable from meteorological data file.

Definition at line 853 of file libtrac.c.

```
00860
00861
         static float help[EX * EY * EP];
00862
00863
        int ip, ix, iv, n = 0, varid;
00864
         /* Check if variable exists... */
00866
         if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00867
          if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00868
             return:
00869
         /* Read data...
00870
00871
         NC(nc_get_var_float(ncid, varid, help));
00872
00873
         /* Copy and check data... */
        for (ip = 0; ip < met->np; ip++)
  for (iy = 0; iy < met->ny; iy++)
00874
00875
             for (ix = 0; ix < met->nx; ix++) {
  dest[ix][iy][ip] = scl * help[n++];
00876
00878
                if (fabs(dest[ix][iy][ip] / scl) > 1e14)
00879
                  dest[ix][iy][ip] = GSL_NAN;
00880
             }
00881 }
```

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

Convert meteorological data from model levels to pressure levels.

Definition at line 885 of file libtrac.c.

```
00888

00899

00890 double aux[EP], p[EP], pt;

00891

00892 int ip, ip2, ix, iy;

00893

00894 /* Loop over columns... */

00895 for (ix = 0; ix < met->nx; ix++)

00896 for (iy = 0; iy < met->ny; iy++) {
```

```
/* Copy pressure profile... */
              for (ip = 0; ip < met->np; ip++)
  p[ip] = met->pl[ix][iy][ip];
00899
00900
00901
              /* Interpolate... */
for (ip = 0; ip < ctl->met_np; ip++) {
   pt = ctl->met_p[ip];
00902
00903
00904
00905
                if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
00906
                  pt = p[0];
                else if ((pt > p[met->np - 1] && p[1] > p[0])
  || (pt < p[met->np - 1] && p[1] < p[0]))
00907
00908
                  pt = p[met->np - 1];
00909
                00910
00911
00912
00913
00914
              /* Copy data... */
for (ip = 0; ip < ctl->met_np; ip++)
00915
00916
00917
                var[ix][iy][ip] = (float) aux[ip];
00918
00919 }
```

Here is the call graph for this function:



5.13.2.23 void read\_met\_periodic ( met\_t \* met )

Create meteorological data with periodic boundary conditions.

Definition at line 923 of file libtrac.c.

```
00924
00925
00926
          int ip, iy;
00928
          /* Check longitudes... */
00929
          if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
00930
                         + met - > lon[1] - met - > lon[0] - 360) < 0.01))
00931
             return:
00932
00933
          /* Increase longitude counter... */
00934
          if ((++met->nx) > EX)
00935
             ERRMSG("Cannot create periodic boundary conditions!");
00936
         /* Set longitude... */
met->lon[met->nx - 2] + met->lon[1] - met->
00937
00938
        lon[0];
00939
00940
           /* Loop over latitudes and pressure levels... */
00941
          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->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
  met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
00942
00943
00944
00945
                met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
00946
00947
               met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
00948
00949
00950
00951 }
```

5.13.2.24 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 955 of file libtrac.c.

```
00962
00963
00964
        FILE *in = NULL;
00965
        char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
00966
          msg[LEN], rvarname[LEN], rval[LEN];
00967
00968
00969
        int contain = 0, i;
00970
00971
         /* Open file... */
        if (filename[strlen(filename) - 1] != '-')
if (!(in = fopen(filename, "r")))
00972
00973
             ERRMSG("Cannot open file!");
00975
00976
        /* Set full variable name... */
        if (arridx >= 0) {
00977
         sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
00978
00979
00980
        } else
         sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
00981
00982
00983
00984
00985
        /* Read data... */
        if (in != NULL)
00986
         while (fgets(line, LEN, in))
00987
00988
            if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
00989
               if (strcasecmp(rvarname, fullname1) == 0 | |
00990
                    strcasecmp(rvarname, fullname2) == 0) {
00991
                  contain = 1;
00992
                 break;
00993
               }
00994
        for (i = 1; i < argc - 1; i++)</pre>
        if (strcasecmp(argv[i], fullname1) == 0 ||
    strcasecmp(argv[i], fullname2) == 0) {
    sprintf(rval, "%s", argv[i + 1]);
00995
00996
00997
00998
             contain = 1;
00999
             break;
01000
01001
        /* Close file... */
if (in != NULL)
01002
01003
01004
          fclose(in);
01006
        /* Check for missing variables... */
01007
        if (!contain) {
        if (strlen(defvalue) > 0)
01008
01009
             sprintf(rval, "%s", defvalue);
01010
          else {
01011
            sprintf(msg, "Missing variable %s!\n", fullname1);
01012
             ERRMSG(msg);
01013
01014 }
01015
        /* Write info... */
01016
        printf("%s = %s\n", fullname1, rval);
01017
01019
         /* Return values... */
01020
        if (value != NULL)
01021
          sprintf(value, "%s", rval);
01022
         return atof(rval);
01023 }
```

5.13.2.25 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 1027 of file libtrac.c.

```
01036
01037
       struct tm t0, t1;
01038
       t0.tm_year = 100;
01039
01040
       t0.tm mon = 0;
        t0.tm_mday = 1;
01041
01042
        t0.tm\_hour = 0;
01043
       t0.tm_min = 0;
01044
       t0.tm_sec = 0;
01045
       t1.tm_year = year - 1900;
01046
       t1.tm_mon = mon - 1;
01047
01048
       t1.tm_mday = day;
01049
       t1.tm_hour = hour;
01050
       t1.tm_min = min;
       t1.tm_sec = sec;
01051
01052
       *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01054 }
```

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

Measure wall-clock time.

Definition at line 1058 of file libtrac.c.

```
01061
                   {
01062
01063
        static double starttime[NTIMER], runtime[NTIMER];
01064
       /* Check id... */
if (id < 0 || id >= NTIMER)
01065
01066
01067
          ERRMSG("Too many timers!");
01068
01069
        /* Start timer... */
        if (mode == 1) {
01070
01071
        if (starttime[id] <= 0)</pre>
01072
           starttime[id] = omp_get_wtime();
          else
01074
            ERRMSG("Timer already started!");
01075
01076
        /* Stop timer... */
else if (mode == 2) {
01077
01078
        if (starttime[id] > 0) {
01080
            runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
01081
            starttime[id] = -1;
01082
            ERRMSG("Timer not started!");
01083
01084
01085
01086
       /* Print timer... */
01087
       else if (mode == 3)
          printf("%s = %g s\n", name, runtime[id]);
01088
01089 }
```

## 5.13.2.27 double tropopause (double t, double lat)

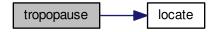
Definition at line 1093 of file libtrac.c.

```
01097
                   static double doys[12]
                   = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01098
01099
01100
                   static double lats[73]
                      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, -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5, 15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5, 45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5, 75, 77.5, 80, 82.5, 85, 87.5, 90
01101
01102
01103
01104
01105
01106
01107
01108
```

```
01110
             static double tps[12][73]
01111
                = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
01112
                         297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
01113
                         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,
01114
01115
                         152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
01116
                         277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
01117
             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,
01118
01119
01120
               150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
01121
               98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
01122
01123
               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,
01124
              284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01125
               287.5, 286.2, 285.8},
01126
             {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,
01129
01130
              100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01131
              99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
              186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8, 279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
01132
01133
               304.3, 304.9, 306, 306.6, 306.2, 306},
01134
             {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
01135
01136
              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,
01137
01138
01139
              148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6,
01140
               263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.
01141
01142
               315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
             {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8, 260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9, 205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7, 101.8, 101.4, 101.1, 101, 101, 101.1, 101.2, 101.5, 101.9,
01143
01144
01145
               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,
01148
01149
              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},
01150
             323.3, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 32
01151
01152
01154
01155
              106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
01156
              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,
01157
             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,
01158
              187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
01160
01161
              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,
01162
01163
              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,
01164
               275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
01166
             {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01167
01168
              185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
01169
              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, 111.6, 111.9, 112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
01170
              120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
01172
              230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7
01173
01174
              278.2, 282.6, 287.4, 290.9, 292.5, 293},
01175
             {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
01176
               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,
01177
              114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4,
               110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
01179
01180
              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},
01181
01182
             {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,
01183
               237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7,
01185
                                                                                                124.8,
01186
               111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01187
              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,
01188
               206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01189
               279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
               305.1},
01191
             {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2
01192
01193
              253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
              223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9, 108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
01194
01195
```

```
102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
01197
             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,
01198
            286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6}, {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7, 284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192, 175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1, 100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
01199
01200
01201
01202
01203
             100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160, 186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278, 280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4, 281.7, 281.1, 281.2}
01204
01205
01206
01207
01208
01209
01210
            double doy, p0, p1, pt;
01211
01212
            int imon, ilat;
01213
            /* Get day of year... */
01214
01215
            doy = fmod(t / 86400., 365.25);
01216
            while (doy < 0)
01217
               doy += 365.25;
01218
01219
            /* Get indices... */
           imon = locate(doys, 12, doy);
ilat = locate(lats, 73, lat);
01220
01221
01222
01223
             /* Get tropopause pressure... */
           01224
01225
            p1 = LIN(lats[ilat], tps[imon + 1][ilat],
lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
01226
01227
01228
            pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01229
01230
            /* Return tropopause pressure... */
01231
            return pt;
01232 }
```

Here is the call graph for this function:



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

Write atmospheric data.

Definition at line 1236 of file libtrac.c.

```
01240
01241
01242
        FILE *in, *out;
01243
01244
        char line[LEN];
01245
01246
        double r, t0, t1;
01247
01248
        int ip, iq, year, mon, day, hour, min, sec;
01249
01250
         /\star Set time interval for output... \star/
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01251
01252
01253
01254
        /* Check if gnuplot output is requested... */
        if (ctl->atm_gpfile[0] != '-') {
```

```
01256
01257
           /* Write info... */
01258
           printf("Plot atmospheric data: %s.png\n", filename);
01259
01260
           /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
01261
01262
             ERRMSG("Cannot create pipe to gnuplot!");
01263
           /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
01264
01265
01266
01267
           /* Set time string... */
           jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01268
01269
01270
                    year, mon, day, hour, min);
01271
           /* Dump gnuplot file to pipe... */
01272
           if (!(in = fopen(ctl->atm_gpfile, "r")))
ERRMSG("Cannot open file!");
01273
01275
           while (fgets(line, LEN, in))
             fprintf(out, "%s", line);
01276
01277
           fclose(in);
01278
01279
01280
        else {
01282
           /\star Write info... \star/
01283
           printf("Write atmospheric data: %s\n", filename);
01284
01285
           /* Create file... */
          if (!(out = fopen(filename, "w")))
01286
01287
             ERRMSG("Cannot create file!");
01288
01289
01290
         /* Write header... */
        fprintf(out,
    "# $1 = time [s]\n"
01291
01292
                  "# $2 = altitude [km] \n"
01294
                  "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
        01295
01296
01297
01298
01299
01300
         /* Write data... */
01301
         for (ip = 0; ip < atm->np; ip++) {
01302
           /* Check time... */
if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
01303
01304
01305
             continue:
01306
01307
           /* Write output... */
           fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
    atm->lon[ip], atm->lat[ip]);
01308
01309
           for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01310
01311
01312
01313
01314
           fprintf(out, "\n");
01315
01316
        /* Close file... */
01317
01318
        fclose(out);
01319 }
```

Here is the call graph for this function:



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

Write CSI data.

Definition at line 1323 of file libtrac.c.

```
01327
                    {
01328
01329
        static FILE *in, *out;
01330
01331
        static char line[LEN]:
01332
        static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01333
          rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01335
01336
        static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01337
01338
         /* Init... */
01339
        if (!init) {
01340
          init = 1;
01341
01342
           /* Check quantity index for mass... */
01343
           if (ctl->qnt_m < 0)
             ERRMSG("Need quantity mass to analyze CSI!");
01344
01345
01346
           /* Open observation data file... */
01347
           printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
01348
           if (!(in = fopen(ctl->csi_obsfile, "r")))
             ERRMSG("Cannot open file!");
01349
01350
01351
           /* Create new file... */
          /* Create new IIIe... ^/
printf("Write CSI data: %s\n", filename);
if (!/out = fonen(filename, "w")))
01352
01353
01354
            ERRMSG("Cannot create file!");
01355
01356
           /* Write header... */
01357
          fprintf(out,
                    "# $1 = time [s]\n"
01358
01359
                   "# $2 = number of hits (cx) \n"
01360
                    "# $3 = number of misses (cy) \n"
01361
                    "# $4 = number of false alarms (cz)\n"
                    "# $5 = number of observations (cx + cy) \n"
01362
                    "# $6 = number of forecasts (cx + cz) \n"
01363
                    "# $7 = bias (forecasts/observations) [%%]\n"
01364
                   "# $8 = probability of detection (POD) [%%]\n"
"# $9 = false alarm rate (FAR) [%%]\n"
01365
01366
01367
                    "# $10 = critical success index (CSI) [%%]\n\n");
01368
01369
01370
        /* Set time interval... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01371
01372
01373
         /\star Initialize grid cells... \star/
01374
        for (ix = 0; ix < ctl->csi_nx; ix++)
  for (iy = 0; iy < ctl->csi_ny; iy++)
    for (iz = 0; iz < ctl->csi_nz; iz++)
01375
01376
01377
01378
               modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01379
01380
         /* Read data... */
01381
        while (fgets(line, LEN, in)) {
01382
01383
           /* Read data... *
           if (sscanf(line, "%lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01384
01385
01386
             continue;
01387
           /* Check time... */
01388
01389
           if (rt < t0)
01390
             continue;
01391
           if (rt > t1)
01392
            break;
01393
01394
           /* Calculate indices... */
          ix = (int) ((rlon - ctl->csi_lon0)
01395
                         / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01396
01397
           iy = (int) ((rlat - ctl -> csi_lat0))
01398
                         / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
           iz = (int) ((rz - ctl -> csi_z0)
01399
01400
                        / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01401
01402
           /* Check indices... */
01403
           if (ix < 0 || ix >= ctl->csi_nx ||
```

```
iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01405
01406
01407
           /* Get mean observation index... */
01408
          obsmean[ix][iy][iz] += robs;
          obscount[ix][iy][iz]++;
01409
01410
01411
01412
        /* Analyze model data... ∗/
01413
        for (ip = 0; ip < atm->np; ip++) {
01414
01415
          /* Check time... */
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
01416
01417
            continue;
01418
          /* Get indices... */
ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01419
01420
                        / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01421
          iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
                        / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
          01424
01425
01426
          /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
01427
01428
              iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01429
01430
01431
01432
          /* Get total mass in grid cell... */
          modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01433
01434
01435
01436
        /* Analyze all grid cells... */
01437
        for (ix = 0; ix < ctl->csi_nx; ix++)
         for (iy = 0; iy < ctl->csi_ny; iy++)
  for (iz = 0; iz < ctl->csi_nz; iz++) {
01438
01439
01440
               /\star Calculate mean observation index... \star/
01442
               if (obscount[ix][iy][iz] > 0)
01443
                obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01444
01445
               /* Calculate column density... */
              if (modmean[ix][iy][iz] > 0) {
  dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
  dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
01446
01447
01448
01449
                 lat = ctl->csi\_lat0 + dlat * (iy + 0.5);
                area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180. * cos(lat * M_PI / 180.);
01450
01451
                modmean[ix][iy][iz] /= (1e6 * area);
01452
01453
01454
01455
               /* Calculate CSI... */
01456
               if (obscount[ix][iy][iz] > 0) {
01457
               if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
                     modmean[ix][iy][iz] >= ctl->csi_modmin)
01458
01459
                   cx++;
                else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
                          modmean[ix][iy][iz] < ctl->csi_modmin)
01461
01462
                 01463
01464
01465
                   cz++;
01466
              }
01467
01468
01469
        /* Write output... */
01470
        if (fmod(t, ctl->csi_dt_out) == 0) {
01471
          /* Write... */
fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
01472
                  (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN, (cx + cz > 0) ? (100. * cx) / (cx + cz) : GSL_NAN,
01474
01475
01476
01477
01478
                   (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01479
01480
          /* Set counters to zero... */
01481
          cx = cy = cz = 0;
01482
01483
        /* Close file... */
01484
        if (t == ctl->t_stop)
01485
          fclose(out);
01486
01487 }
```

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

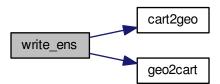
Write ensemble data.

Definition at line 1491 of file libtrac.c.

```
01495
                     {
01496
01497
        static FILE *out;
01498
        static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
01499
          t0, t1, x[NENS][3], xm[3];
01500
01501
        static int init, ip, iq;
01503
01504
        static size_t i, n;
01505
01506
         /* Init... */
01507
         if (!init) {
01508
          init = 1;
01510
           /* Check quantities... */
           if (ctl->qnt_ens < 0)
    ERRMSG("Missing ensemble IDs!");</pre>
01511
01512
01513
01514
           /* Create new file... */
           printf("Write ensemble data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01515
01516
             ERRMSG("Cannot create file!");
01517
01518
01519
           /* Write header... */
01520
           fprintf(out,
01521
                     "# $1 = time [s] \n"
                    "# \$2 = altitude [km]\n"
"# \$3 = longitude [deg]\n" "# \$4 = latitude [deg]\n");
01522
01523
           01524
01525
01526
           01527
01528
01529
           fprintf(out, "# \$%d = number of membersn, 5 + 2 * ctl->nq);
01530
        }
01531
01532
01533
         /* Set time interval... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01534
01535
01536
         /* Init... */
01537
01538
        ens = GSL_NAN;
        n = 0;
01539
01540
         /* Loop over air parcels... */
for (ip = 0; ip < atm->np; ip++) {
01541
01542
01543
01544
           /* Check time... */
01545
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
01546
            continue;
01547
01548
           /\star Check ensemble id... \star/
           if (atm->q[ctl->qnt_ens][ip] != ens) {
01549
01550
01551
             /* Write results... */
             if (n > 0) {
01553
01554
                /* Get mean position... */
                for (i = 0; i < n; i++) {
    xm[0] += x[i][0] / (double) n;
    xm[1] += x[i][1] / (double) n;
    xm[2] += x[i][2] / (double) n;</pre>
01555
01556
01557
01558
01559
01560
                cart2geo(xm, &dummy, &lon, &lat);
fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
01561
01562
01563
                        lat);
01564
01565
                /* Get quantity statistics... */
                for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01566
01567
01568
                  fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01569
                for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01570
01571
```

```
fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01573
01574
                 fprintf(out, " %lu\n", n);
             }
01575
01576
01577
              /* Init new ensemble... */
01578
              ens = atm->q[ctl->qnt_ens][ip];
01579
              n = 0;
01580
01581
01582
           /* Save data... */
01583
           p[n] = atm->p[ip];
01584
            geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
01585
           for (iq = 0; iq < ctl->nq; iq++)
           q[iq][n] = atm->q[iq][ip];
if ((++n) >= NENS)
    ERRMSG("Too many data points!");
01586
01587
01588
01589
01590
01591
         /* Write results... */
01592
         if (n > 0) {
01593
01594
            /* Get mean position... */
           xm[0] = xm[1] = xm[2] = 0;
for (i = 0; i < n; i++) {
  xm[0] += x[i][0] / (double) n;
  xm[1] += x[i][1] / (double) n;</pre>
01595
01596
01597
01598
             xm[2] += x[i][2] / (double) n;
01599
01600
           cart2geo(xm, &dummy, &lon, &lat);
fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
01601
01602
01603
01604
            /* Get quantity statistics... */
           for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01605
01606
01607
              fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01608
           for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01609
01610
01611
              fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01612
           fprintf(out, " %lu\n", n);
01613
01614
01615
01616
         /* Close file... */
01617
         if (t == ctl->t_stop)
01618
           fclose(out);
01619 }
```

Here is the call graph for this function:



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

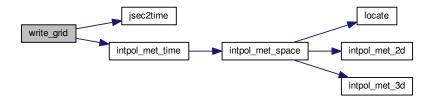
Write gridded data.

Definition at line 1623 of file libtrac.c.

```
{
01630
01631
         FILE *in, *out;
01632
01633
         char line[LEN1:
01634
01635
         static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01636
            area, rho_air, press, temp, cd, mmr, t0, t1, r;
01637
01638
         static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01639
01640
         /* Check dimensions... */
         if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
    ERRMSG("Grid dimensions too large!");
01641
01642
01643
01644
          /\star Check quantity index for mass... \star/
         if (ctl->qnt_m < 0)
01645
           ERRMSG("Need quantity mass to write grid data!");
01646
01647
01648
         /* Set time interval for output... */
         t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01649
01650
01651
         /* Set grid box size... */
dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
01652
01653
01654
01655
01656
01657
          /* Initialize grid... */
01658
         for (ix = 0; ix < ctl->grid_nx; ix++)
          for (iy = 0; iy < ctl->grid_ny; iy++)
    for (iz = 0; iz < ctl->grid_nz; iz++)
01659
01660
01661
                 grid_m[ix][iy][iz] = 0;
01662
         /* Average data... */
for (ip = 0; ip < atm->np; ip++)
   if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
01663
01664
01665
01666
01667
              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);
01668
01669
01670
01671
01672
               /* Check indices... */
              if (ix < 0 || ix >= ctl->grid_nx ||
01673
01674
                   iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01675
                 continue;
01676
              /* Add mass... */
grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01677
01678
01679
01680
01681
         /\star Check if gnuplot output is requested... \star/
01682
         if (ctl->grid_gpfile[0] != '-') {
01683
01684
            /* Write info... */
            printf("Plot grid data: %s.png\n", filename);
01685
01686
            /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
01687
01688
              ERRMSG("Cannot create pipe to gnuplot!");
01689
01690
            /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
01691
01692
01693
01694
            /* Set time string... */
01695
            01696
01697
                     year, mon, day, hour, min);
01699
            /\star Dump gnuplot file to pipe... \star/
            if (!(in = fopen(ctl->grid_gpfile, "r")))
01700
              ERRMSG("Cannot open file!");
01701
            while (fgets(line, LEN, in))
fprintf(out, "%s", line);
01702
01703
01704
            fclose(in);
01705
01706
01707
         else {
01708
01709
            /* Write info... */
            printf("Write grid data: %s\n", filename);
01711
            /* Create file... */
01712
01713
            if (!(out = fopen(filename, "w")))
              ERRMSG("Cannot create file!");
01714
01715
```

```
01716
         /* Write header... */
01717
01718
         fprintf(out,
                  "# $1 = time [s] \n"
01719
                  "# $2 = altitude [km] \n"
01720
                  "# $3 = longitude [deg]\n"
01721
                  "# $4 = latitude [deg]\n"
01723
                  "# $5 = surface area [km^2]\n"
01724
                  "# $6 = layer width [km] \n"
                  "# \$7 = temperature [K]\n"
01725
                  "# $8 = column density [kg/m^2]\n"
01726
01727
                  "# $9 = mass mixing ratio [1]\n\n");
01728
         /* Write data... */
01729
01730
         for (ix = 0; ix < ctl->grid_nx; ix++) {
          if (ix > 0 && ctl->grid_nx, ix++) {
   if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
   fprintf(out, "\n");
   for (iy = 0; iy < ctl->grid_ny; iy++) {
      if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
01731
01732
01733
01734
                fprintf(out, "\n");
              for (iz = 0; iz < ctl->grid_nz; iz++)
01736
01737
                if (!ctl->grid_sparse
                     || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) 
01738
01739
01740
                  /* Set coordinates... */
01741
                  z = ctl->grid_z0 + dz * (iz + 0.5);
01742
                  lon = ctl - > grid_lon0 + dlon * (ix + 0.5);
                  lat = ctl - > grid_lat0 + dlat * (iy + 0.5);
01743
01744
01745
                  /\star Get pressure and temperature... \star/
01746
                  press = P(z);
01747
                  intpol_met_time(met0, met1, t, press, lon, lat,
01748
                                     NULL, &temp, NULL, NULL, NULL, NULL, NULL);
01749
01750
                  /* Calculate surface area... */
                  area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
 * cos(lat * M_PI / 180.);
01751
01752
01753
01754
                   /* Calculate column density... */
01755
                  cd = grid_m[ix][iy][iz] / (1e6 * area);
01756
                  /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
01757
01758
                  mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01759
01760
                   /* Write output... */
01761
01762
                  fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
01763
                            t, z, lon, lat, area, dz, temp, cd, mmr);
01764
                }
01765
           }
01766
        }
01767
01768
        /* Close file... */
01769
        fclose(out);
01770 }
```

Here is the call graph for this function:



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

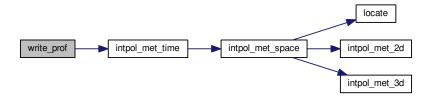
Write profile data.

Definition at line 1774 of file libtrac.c.

```
{
01781
01782
         static FILE *in, *out;
01783
01784
         static char line[LEN];
01785
        static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY], rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z,
01786
01787
01788
           press, temp, rho_air, mmr, h2o, o3;
01789
01790
        static int init, obscount[GX][GY], ip, ix, iy, iz;
01791
01792
         /* Init... */
01793
         if (!init) {
01794
           init = 1;
01795
01796
           /\star Check quantity index for mass... \star/
           if (ctl->qnt_m < 0)
01797
              ERRMSG("Need quantity mass!");
01798
01799
01800
            /* Check dimensions...
           if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
01801
             ERRMSG("Grid dimensions too large!");
01802
01803
01804
            /* Open observation data file... */
           printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01805
01806
            if (!(in = fopen(ctl->prof_obsfile, "r")))
01807
              ERRMSG("Cannot open file!");
01808
01809
           /* Create new file... */
           printf("Write profile data: %s\n", filename);
if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");
01810
01811
01812
01813
01814
            /* Write header... */
           fprintf(out,
    "# $1
01815
                            = time [s]\n"
01816
                            = altitude [km]\n"
01817
01818
                     "# $3
                            = longitude [deg]\n"
01819
                     "# $4
                            = latitude [deg]\n"
                            = pressure [hPa]\n"
01820
                     "# $5
                     "# $6 = temperature [K] \n"
01821
                     "# $7 = mass mixing ratio [1]\n"
01822
01823
                     "# $8 = H2O volume mixing ratio [1]\n"
                     "# $9 = O3 volume mixing ratio [1]\n"
01824
01825
                     "# $10 = mean BT index [K] \n");
01826
01827
           /* 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;
01828
01829
01830
01831
01832
        /* Set time interval... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01833
01834
01835
01837
         /* Initialize... */
01838
         for (ix = 0; ix < ctl->prof_nx; ix++)
01839
           for (iy = 0; iy < ctl->prof_ny; iy++) {
              obsmean[ix][iy] = 0;
01840
              obscount[ix][iy] = 0;
01841
              tmean[ix][iy] = 0;
for (iz = 0; iz < ctl->prof_nz; iz++)
01842
01843
01844
                mass[ix][iy][iz] = 0;
01845
01846
         /* Read data... */
01847
         while (fgets(line, LEN, in)) {
01848
01850
            /* Read data... */
           if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01851
01852
             continue;
01853
           /* Check time... */
01854
01855
           if (rt < t0)
01856
              continue;
01857
           if (rt > t1)
01858
             break;
01859
           /* Calculate indices... */
01860
           ix = (int) ((rlon - ctl->prof_lon0) / dlon);
01861
01862
           iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01863
           /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01864
01865
01866
             continue;
```

```
01867
01868
             /* Get mean observation index... */
            obsmean[ix][iy] += robs;
tmean[ix][iy] += rt;
01869
01870
01871
            obscount[ix][iy]++;
01872
01873
01874
          /★ Analyze model data... ★/
01875
          for (ip = 0; ip < atm->np; ip++) {
01876
01877
             /\star Check time... \star/
            if (atm->time[ip] < t0 || atm->time[ip] > t1)
01878
01879
              continue;
01880
01881
            /* 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);
01882
01883
01884
01885
            /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx ||
01886
01887
                 iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01888
01889
               continue;
01890
01891
             /* Get total mass in grid cell... */
            mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01892
01893
01894
01895
          /* Extract profiles... */
          for (ix = 0; ix < ctl->prof_nx; ix++)
  for (iy = 0; iy < ctl->prof_ny; iy++)
01896
01897
01898
              if (obscount[ix][iy] > 0) {
01899
01900
                 /* Write output... */
01901
                 fprintf(out, "\n");
01902
                  /* Loop over altitudes... */
01903
                 for (iz = 0; iz < ctl->prof_nz; iz++) {
01904
01905
01906
                    /* 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);
01907
01908
01909
01910
01911
                    /* Get meteorological data... */
01912
                    press = P(z);
01913
                    intpol_met_time(met0, met1, t, press, lon, lat,
                                        NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
01914
01915
                   /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
* cos(lat * M_PI / 180.);
01916
01917
01918
01919
01920
                   mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
01921
                    /* Write output... */
01922
                    fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
01924
                              tmean[ix][iy] / obscount[ix][iy],
01925
                              z, lon, lat, press, temp, mmr, h2o, o3,
                              obsmean[ix][iy] / obscount[ix][iy]);
01926
01927
01928
01929
01930
         /* Close file... */
01931
          if (t == ctl->t_stop)
01932
            fclose(out);
01933 }
```

Here is the call graph for this function:



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

Write station data.

Definition at line 1937 of file libtrac.c.

```
01941
                   {
01942
01943
        static FILE *out;
01945
        static double rmax2, t0, t1, x0[3], x1[3];
01946
01947
        static int init, ip, iq;
01948
01949
        /* Init... */
01950
        if (!init) {
01951
          init = 1;
01952
01953
          /* Write info... */
          printf("Write station data: %s\n", filename);
01954
01955
01956
          /* Create new file... */
01957
          if (!(out = fopen(filename, "w")))
01958
            ERRMSG("Cannot create file!");
01959
          /* Write header... */
01960
01961
          fprintf(out,
                   "# $1 = time [s]\n"
"# $2 = altitude [km]\n"
01962
01963
01964
                   "# $3 = longitude [deg] \n" "# $4 = latitude [deg] \n");
          01965
01966
01967
01968
          fprintf(out, "\n");
01969
01970
           /* Set geolocation and search radius... */
01971
          geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
01972
          rmax2 = gsl_pow_2(ctl->stat_r);
01973
01974
01975
        /* Set time interval for output... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01976
01977
01978
01979
        /* Loop over air parcels... */
for (ip = 0; ip < atm->np; ip++) {
01980
01981
01982
           /* Check time... */
01983
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
01984
             continue;
01985
          /* Check station flag... */
if (ctl->qnt_stat >= 0)
01986
01987
01988
            if (atm->q[ctl->qnt_stat][ip])
01989
01990
01991
          /* Get Cartesian coordinates... */
          geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
01992
01993
01994
          /\star Check horizontal distance... \star/
```

5.14 libtrac.h 125

```
if (DIST2(x0, x1) > rmax2)
01996
            continue;
01997
          /* Set station flag... */
if (ctl->qnt_stat >= 0)
atm->q[ctl->qnt_stat][ip] = 1;
01998
01999
02000
02002
           /* Write data... */
          02003
02004
          for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02005
02006
02007
            fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02008
02009
          fprintf(out, "\n");
02010
02011
        /* Close file... */
if (t == ctl->t_stop)
02012
02013
02014
          fclose(out);
02015 }
```

Here is the call graph for this function:



# 5.14 libtrac.h

```
00001 /*
00002
        This file is part of MPTRAC.
00003
00004
        \ensuremath{\mathsf{MPTRAC}} is free software: you can redistribute it and/or modify
00005
        it under the terms of the GNU General Public License as published by
00006
        the Free Software Foundation, either version 3 of the License, or
00007
        (at your option) any later version.
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
        Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00034 #include <ctype.h>
00035 #include <gsl/gsl_const_mksa.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_statistics.h>
00041 #include <math.h>
00042 #include <netcdf.h>
00043 #include <omp.h>
00044 #include <stdio.h>
00045 #include <stdlib.h>
00046 #include <string.h>
00047 #include <time.h>
00048 #include <sys/time.h>
00049
00050 /* -
00051
         Macros...
00052
00053
```

```
00055 #define ALLOC(ptr, type, n)
      if((ptr=calloc((size_t)(n), sizeof(type)))==NULL)
00056
00057
          ERRMSG("Out of memory!");
00058
00060 #define DIST(a, b) sqrt(DIST2(a, b))
00061
00063 #define DIST2(a, b)
        ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))
00064
00065
00067 #define DOTP(a, b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
00068
00070 #define ERRMSG(msg) {
        printf("\nError (%s, %s, 1%d): %s\n\n",
00071
00072
                    __FILE__, __func__, __LINE__, msg);
00073
          exit(EXIT_FAILURE);
00074
00075
00077 #define LIN(x0, y0, x1, y1, x)
00078 ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))
00079
00081 #define NC(cmd) {
00082
        if((cmd)!=NC_NOERR)
            ERRMSG(nc_strerror(cmd));
00083
00084
00085
00087 #define NORM(a) sqrt(DOTP(a, a))
88000
00090 #define PRINT(format, var)
00091 printf("Print (%s, %s, 1%d): %s= "format"\n",
00092 __FILE__, _func__, _LINE__, #var, var);
00093
00095 #define P(z) (P0 \times exp(-(z)/H0))
00096
00098 #define TOK(line, tok, format, var) {
00099     if(((tok)=strtok((line), " \t"))) {
00100     if(sscanf(tok, format, &(var))!=1) continue;
          } else ERRMSG("Error while reading!");
00101
00102
00103
00105 #define Z(p) (H0*log(P0/(p)))
00106
00108 #define START TIMER(id) timer(#id, id, 1)
00109
00111 #define STOP_TIMER(id) timer(#id, id, 2)
00114 #define PRINT_TIMER(id) timer(#id, id, 3)
00115
00116 /* --
         Constants...
00117
00118
00119
00121 #define G0 9.80665
00122
00124 #define H0 7.0
00125
00127 #define P0 1013.25
00130 #define RE 6367.421
00131
00132 /* -----
00133
         Dimensions...
00134
00135
00137 #define LEN 5000
00138
00140 #define NP 10000000
00141
00143 #define NO 10
00144
00146 #define EP 73
00147
00149 #define EX 721
00150
00152 #define EY 361
00153
00155 #define GX 720
00156
00158 #define GY 360
00159
00161 #define GZ 100
00162
00164 #define NENS 2000
00165
00167 #define NTHREADS 128
00168
00170 #define NTIMER 20
00171
```

5.14 libtrac.h 127

```
00172 /*
00173
         Structs...
00174
00175
00177 typedef struct {
00178
00180
        int nq;
00181
00183
        char qnt_name[NQ][LEN];
00184
        char qnt_unit[NQ][LEN];
00186
00187
00189
        char gnt format[NO][LEN];
00190
00192
        int qnt_ens;
00193
00195
        int qnt_m;
00196
00198
        int qnt_rho;
00199
00201
        int qnt_r;
00202
00204
        int qnt_ps;
00205
00207
        int qnt_p;
00208
00210
        int qnt_t;
00211
00213
        int qnt_u;
00214
00216
        int qnt_v;
00217
00219
        int qnt_w;
00220
00222
        int qnt_h2o;
00223
00225
        int qnt_o3;
00226
00228
        int qnt_theta;
00229
00231
        int qnt_pv;
00232
00234
        int qnt_tice;
00235
        int qnt_tsts;
00238
00240
        int qnt_tnat;
00241
00243
        int qnt_stat;
00244
00246
        int direction;
00247
00249
        double t_start;
00250
00252
        double t_stop;
00253
        double dt_mod;
00256
00258
        double dt_met;
00259
00261
        int met_np;
00262
00264
        double met_p[EP];
00265
00268
        int isosurf;
00269
00271
        char balloon[LEN];
00272
00274
        double turb dx trop;
00275
00277
        double turb_dx_strat;
00278
00280
        double turb_dz_trop;
00281
00283
        double turb dz strat;
00284
00286
        double turb_meso;
00287
00289
        double tdec_trop;
00290
00292
        double tdec_strat;
00293
00295
        double psc_h2o;
00296
00298
        double psc_hno3;
00299
00301
        char atm basename[LEN];
```

```
00302
00304
        char atm_gpfile[LEN];
00305
00307
        double atm_dt_out;
00308
        int atm_filter;
00310
00311
00313
        char csi_basename[LEN];
00314
00316
        double csi_dt_out;
00317
00319
        char csi_obsfile[LEN];
00320
00322
        double csi_obsmin;
00323
00325
00326
        double csi_modmin;
00328
        int csi nz;
00329
00331
        double csi_z0;
00332
00334
        double csi_z1;
00335
00337
        int csi nx;
00338
00340
        double csi_lon0;
00341
00343
        double csi_lon1;
00344
00346
        int csi_ny;
00347
00349
        double csi_lat0;
00350
00352
        double csi_lat1;
00353
        char grid_basename[LEN];
00355
00356
        char grid_gpfile[LEN];
00359
00361
        double grid_dt_out;
00362
        int grid_sparse;
00364
00365
00367
        int grid_nz;
00368
00370
        double grid_z0;
00371
00373
        double grid_z1;
00374
00376
        int grid_nx;
00377
00379
        double grid_lon0;
00380
00382
        double grid_lon1;
00383
00385
        int grid_ny;
00386
00388
        double grid_lat0;
00389
00391
        double grid_lat1;
00392
00394
        char prof_basename[LEN];
00395
00397
        char prof_obsfile[LEN];
00398
00400
        int prof_nz;
00401
        double prof_z0;
00403
00404
00406
        double prof_z1;
00407
00409
        int prof_nx;
00410
00412
        double prof_lon0;
00413
00415
        double prof_lon1;
00416
00418
        int prof_ny;
00419
00421
        double prof_lat0;
00422
00424
        double prof_lat1;
00425
00427
        char ens_basename[LEN];
00428
00430
        char stat_basename[LEN];
00431
```

5.14 libtrac.h 129

```
00433
       double stat_lon;
00434
00436
        double stat_lat;
00437
        double stat_r;
00439
00440
00441 } ctl_t;
00442
00444 typedef struct {
00445
00447
        int np;
00448
00450
        double time[NP];
00451
00453
        double p[NP];
00454
        double lon[NP];
00456
00457
00459
        double lat[NP];
00460
00462
        double q[NQ][NP];
00463
00465
        double up[NP];
00466
00468
        double vp[NP];
00469
00471
        double wp[NP];
00472
00473 } atm_t;
00474
00476 typedef struct {
00477
00479
        double time;
00480
00482
        int nx;
00483
00485
        int ny;
00486
00488
        int np;
00489
00491
        double lon[EX];
00492
00494
        double lat[EY];
00495
00497
        double p[EP];
00498
00500
        double ps[EX][EY];
00501
        float pl[EX][EY][EP];
00503
00504
00506
        float t[EX][EY][EP];
00507
00509
        float u[EX][EY][EP];
00510
00512
        float v[EX][EY][EP];
00513
        float w[EX][EY][EP];
00516
00518
        float h2o[EX][EY][EP];
00519
00521
        float o3[EX][EY][EP];
00522
00523 } met_t;
00524
00525 /* --
00526
         Functions...
00527
00528
00530 void cart2geo(
00531
       double *x,
00532
        double *z,
00533
        double *lon,
00534
        double *lat);
00535
00537 double deg2dx(
00538
       double dlon,
00539
        double lat);
00540
00542 double deg2dy(
00543
        double dlat);
00544
00546 double dp2dz(
00547
       double dp,
00548
        double p);
00549
00551 double dx2deg(
00552
       double dx.
```

```
double lat);
00554
00556 double dy2deg(
00557
        double dy);
00558
00560 double dz2dp(
00561 double dz,
00562
        double p);
00563
00565 void geo2cart(
        double z,
00566
        double lon,
00567
00568
        double lat,
00569
        double *x);
00570
00572 void get_met(
00573
        ctl_t * ctl,
00574
        char *metbase,
00575
        double t,
        met_t * met0,
met_t * met1);
00576
00577
00578
00580 void get_met_help(
00581
       double t,
00582
        int direct,
00583
        char *metbase,
00584
        double dt_met,
00585
        char *filename);
00586
00588 void intpol_met_2d(
       double array[EX][EY],
00589
00590
        int ix,
00591
        int iy,
00592
        double wx,
00593
        double wy,
00594
        double *var);
00595
00597 void intpol_met_3d(
00598
        float array[EX][EY][EP],
00599
        int ip,
00600
        int ix,
00601
        int iy,
00602
        double wp,
00603
        double wx,
00604
        double wy,
00605
        double *var);
00606
00612
        double lat,
00613
        double *ps,
00614
        double *t,
00615
        double *u,
00616
        double *v,
00617
        double *w,
00618
        double *h2o,
00619
        double *o3);
00620
00622 void intpol_met_time(
       met_t * met0,
met_t * met1,
00623
00624
00625
        double ts,
00626
        double p,
00627
        double lon,
00628
        double lat,
00629
        double *ps,
        double *t,
00630
00631
        double *u,
00632
        double *v,
00633
        double *w,
00634
        double *h2o,
00635
        double *o3);
00636
00638 void jsec2time(
00639
        double jsec,
00640
        int *year,
00641
        int *mon,
00642
        int *day,
        int *hour,
00643
00644
        int *min,
00645
        int *sec,
00646
        double *remain);
00647
00649 int locate(
00650
        double *xx,
```

5.14 libtrac.h 131

```
00651
        int n,
00652
        double x);
00653
00655 void read_atm(
       const char *filename,
ctl_t * ctl,
00656
00657
        atm_t * atm);
00658
00659
00661 void read_ctl(
00662
       const char *filename,
00663
        int argc,
00664
        char *argv[],
        ctl_t * ctl);
00665
00666
00668 void read_met(
00669 ctl_t * ctl,
00670 char *filename,
00671
       met_t * met);
00672
00674 void read_met_extrapolate(
00675
      met_t * met);
00676
00678 void read_met_help(
00679
       int ncid,
00680
       char *varname,
char *varname2,
00682
        met_t * met,
00683
        float dest[EX][EY][EP],
00684
        float scl);
00685
00687 void read_met_ml2pl(
       ctl_t * ctl,
met_t * met,
00688
00689
00690
        float var[EX][EY][EP]);
00691
00693 void read_met_periodic(
00694
       met_t * met);
00695
00697 double scan_ctl(
00698
       const char *filename,
00699
        int argc,
        char *argv[],
const char *varname,
00700
00701
00702
        int arridx,
00703
        const char *defvalue,
00704
        char *value);
00705
00707 void time2jsec(
00708
       int year,
00709
        int mon.
00710
        int day,
00711
        int hour,
00712
        int min,
00713
        int sec,
00714
        double remain,
00715
        double *jsec);
00716
00718 void timer(
00719 const char *name,
00720
        int id,
00721
       int mode);
00722
00723 /* Get tropopause pressure... */
00724 double tropopause(
00725
       double t,
00726
       double lat);
00727
00729 void write_atm(
00730 const char *filename,
       ctl_t * ctl,
atm_t * atm,
00731
00732
00733
        double t);
00734
00736 void write_csi(
        const char *filename,
00737
00738
        ctl_t * ctl,
00739
        atm_t * atm,
00740
        double t);
00741
00743 void write ens(
00744
       const char *filename,
        ctl_t * ctl,
atm_t * atm,
00745
00746
00747
        double t);
00748
00750 void write_grid(
       const char *filename,
00751
```

```
ctl_t * ctl,
met_t * met0,
met_t * met1,
atm_t * atm,
00752
00753
00754
00755
00756
         double t);
00757
00759 void write_prof(
00760
         const char *filename,
         ctl_t * ctl,
met_t * met0,
00761
00762
         met_t * met1,
atm_t * atm,
00763
00764
00765
         double t);
00766
00768 void write_station(
00769
00770
         const char *filename,
         ctl_t * ctl,
atm_t * atm,
00771
         double t);
```

## 5.15 match.c File Reference

Calculate deviations between two trajectories.

#### **Functions**

• int main (int argc, char \*argv[])

## 5.15.1 Detailed Description

Calculate deviations between two trajectories.

Definition in file match.c.

### 5.15.2 Function Documentation

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

Definition at line 28 of file match.c.

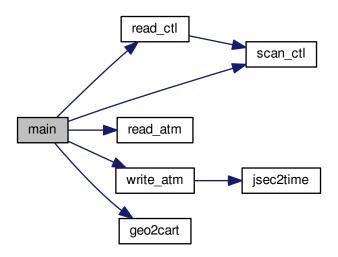
```
00030
00031
00032
        ctl_t ctl;
00033
00034
        atm_t *atm1, *atm2, *atm3;
00035
00036
        FILE *out;
00037
00038
        char filename[LEN];
00039
00040
        double filter_dt, x1[3], x2[3], dh, dq[NQ], dv, 1h = 0, 1t = 0, 1v = 0;
00041
00042
        int filter, ip1, ip2, iq, n;
00043
        /* Allocate... */
00044
        ALLOC(atm1, atm_t, 1);
ALLOC(atm2, atm_t, 1);
ALLOC(atm3, atm_t, 1);
00045
00046
00047
00048
00049
        /* Check arguments... */
00050
00051
        if (argc < 5)
          ERRMSG("Give parameters: <ctl> <atm_test> <atm_ref> <outfile>");
00052
00053
        /* Read control parameters... */
00054
        read_ctl(argv[1], argc, argv, &ctl);
```

```
filter = (int) scan_ctl(argv[1], argc, argv, "FILTER", -1, "0", NULL);
00056
        filter_dt = scan_ctl(argv[1], argc, argv, "FILTER_DT", -1, "0", NULL);
00057
00058
        /* Read atmospheric data... */
00059
        read_atm(argv[2], &ctl, atml);
00060
        read atm(argv[3], &ctl, atm2);
00062
00063
        printf("Write transport deviations: %s\n", argv[4]);
00064
00065
        /* Create output file... */
        if (!(out = fopen(argv[4], "w")))
00066
           ERRMSG("Cannot create file!");
00067
00068
00069
        /* Write header... */
00070
        fprintf(out,
                  "# $1 = time [s] \n"
00071
00072
                  "# $2 = altitude [km] \n"
                  "# \$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],</pre>
00074
00075
00076
                    ctl.qnt_unit[iq]);
        fprintf(out,
    "# $%d = trajectory time [s]\n"
    "# $%d = vertical length of trajectory [km]\n"
    "# $%d = vertical length of trajectory [km]\n"
00077
00078
00079
                  "# $%d = horizontal length of trajectory [km]\n"
00081
                  "# \$%d = vertical deviation [km]\n"
00082
                  "# \$%d = horizontal deviation [km]\n",
00083
                 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
00085
        ctl.qnt_name[iq], ctl.qnt_unit[iq]);
fprintf(out, "\n");
00086
00087
00088
00089
         /\star Filtering of reference time series... \star/
        if (filter) {
00090
00091
           /* Copy data... */
00093
          memcpy(atm3, atm2, sizeof(atm_t));
00094
00095
           /\star Loop over data points... \star/
00096
           for (ip1 = 0; ip1 < atm2->np; ip1++) {
00097
            n = 0:
00098
             atm2->p[ip1] = 0;
00099
             for (iq = 0; iq < ctl.nq; iq++)</pre>
00100
               atm2->q[iq][ip1] = 0;
00101
             for (ip2 = 0; ip2 < atm2->np; ip2++)
               if (fabs(atm2->time[ip1] - atm2->time[ip2]) < filter_dt) {
  atm2->p[ip1] += atm3->p[ip2];
00102
00103
                  for (iq = 0; iq < ctl.nq; iq++)
00104
                   atm2->q[iq][ip1] += atm3->q[iq][ip2];
00105
00106
00107
             atm2->p[ip1] /= n;
for (iq = 0; iq < ctl.nq; iq++)
00108
00109
               atm2->q[iq][ip1] /= n;
00110
00112
           /* Write filtered data... */
sprintf(filename, "%s.filt", argv[3]);
00113
00114
00115
          write_atm(filename, &ctl, atm2, 0);
00116
00117
00118
         /* Loop over air parcels (reference data)... */
00119
         for (ip2 = 0; ip2 < atm2->np; ip2++) {
00120
00121
           /* Get trajectory length... */
00122
           if (ip2 > 0) {
00123
             geo2cart(0, atm2->lon[ip2 - 1], atm2->lat[ip2 - 1], x1);
             geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00125
             lh += DIST(x1, x2);
             lv += fabs(Z(atm2->p[ip2 - 1]) - Z(atm2->p[ip2]));
lt = fabs(atm2->time[ip2] - atm2->time[0]);
00126
00127
00128
00129
00130
           /* Init... */
00131
           n = 0;
00132
           dh = 0;
           dv = 0;
00133
           for (iq = 0; iq < ctl.nq; iq++)</pre>
00134
            dq[iq] = 0;
00135
00136
           geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00137
00138
           /\star Find corresponding time step (test data)... \star/
00139
           for (ip1 = 0; ip1 < atm1->np; ip1++)
             00140
00141
```

```
00142
00143
                  /* Calculate deviations... */
                  geo2cart(0, atm1->lon[ip1], atm1->lat[ip1], x1);
00144
                  dh += DIST(x1, x2);
00145
                  \label{eq:dv += Z(atm1->p[ip1]) - Z(atm2->p[ip2]);} dv += Z(atm1->p[ip1]);
00146
                 for (iq = 0; iq < ctl.nq; iq++)
    dq[iq] += atml->q[iq][ip1] - atm2->q[iq][ip2];
00147
00148
00149
00150
00151
            /* Write output... */
00152
            if (n > 0) {
00153
              fprintf(out, "%.2f %.4f %.4f %.4f",
00154
                        atm2->time[ip2], Z(atm2->p[ip2]),
atm2->lon[ip2], atm2->lat[ip2]);
00155
00156
               for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl.qnt_format[iq], atm2->q[iq][ip2]);
00157
00158
00159
00160
               fprintf(out, " %.2f %g %g %g %g", lt, lv, lh, dv / n, dh / n);
for (iq = 0; iq < ctl.nq; iq++) {
   fprintf(out, " ");</pre>
00161
00162
00163
                 fprintf(out, ctl.qnt_format[iq], dq[iq] / n);
00164
00165
00166
               fprintf(out, "\n");
00167
00168
00169
         /* Close file... */
00170
00171
         fclose(out);
00172
00173
          /* Free... */
00174
         free(atm1);
00175
         free(atm2);
00176
         free(atm3);
00177
00178
         return EXIT_SUCCESS;
00179 }
```

Here is the call graph for this function:



#### 5.16 match.c

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

5.16 match.c 135

```
MPTRAC is free software: you can redistribute it and/or modify
                it under the terms of the GNU General Public License as published by
00005
00006
               the Free Software Foundation, either version 3 of the License, or
00007
                (at your option) any later version.
00008
00009
               MPTRAC is distributed in the hope that it will be useful,
               but WITHOUT ANY WARRANTY; without even the implied warranty of
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
               Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00017
00018 */
00019
00025 #include "libtrac.h"
00026 #include <gsl/gsl_sort.h>
00027
00028 int main(
00029
              int argc,
00030
              char *argv[]) {
00031
00032
              ctl t ctl;
00033
00034
              atm_t *atm1, *atm2, *atm3;
00035
00036
              FILE *out:
00037
00038
               char filename[LEN]:
00039
00040
               double filter_dt, x1[3], x2[3], dh, dq[NQ], dv, lh = 0, lt = 0, lv = 0;
00041
00042
               int filter, ip1, ip2, iq, n;
00043
                /* Allocate... */
00044
               ALLOC(atm1, atm_t, 1);
ALLOC(atm2, atm_t, 1);
00045
00046
00047
               ALLOC(atm3, atm_t, 1);
00048
00049
                /* Check arguments... */
               if (argc < 5)
00050
00051
                  ERRMSG("Give parameters: <ctl> <atm test> <atm ref> <outfile>");
00052
00053
                /* Read control parameters... */
00054
                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);
00055
00056
00057
00058
               /* Read atmospheric data... */
00059
               read_atm(argv[2], &ctl, atm1);
00060
               read_atm(argv[3], &ctl, atm2);
00061
00062
               /* Write info... */
00063
               printf("Write transport deviations: sn'', argv[4]);
00064
00065
                /* Create output file... */
               if (!(out = fopen(argv[4], "w")))
00066
00067
                   ERRMSG("Cannot create file!");
00068
               /* Write header... */
00069
00070
               fprintf(out,
00071
                                "# $1 = time [s] \n"
00072
                                "# $2 = altitude [km] \n"
                                "# $3 = longitude [deg] \n" "# <math>$4 = latitude [deg] \n");
00073
               for (iq = 0; iq < ctl.nq; iq++)
  fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],</pre>
00074
00075
                                   ctl.qnt_unit[iq]);
00076
00077
               fprintf(out.
                                "# \$%d = trajectory time [s]\n"
00078
00079
                                "# \$%d = vertical length of trajectory [km]\n"
                                "# \$%d = horizontal length of trajectory [km]\n"
00080
                                "# \$%d = vertical deviation [km]\n"
00081
                                "# \$%d = horizontal deviation [km]\n",
00082
00083
                                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
00085
               \label{eq:continuity} \begin{array}{cccc} & & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & \\ & & & \\ & & & \\ & & & \\ & & & \\ & & & \\ & & \\ & & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & 
00086
00087
00088
00089
                /\star Filtering of reference time series... \star/
00090
                if (filter) {
00091
00092
                    /* Copy data... */
00093
                   memcpy(atm3, atm2, sizeof(atm_t));
00094
00095
                    /* Loop over data points... */
```

```
for (ip1 = 0; ip1 < atm2->np; ip1++) {
00097
00098
              atm2->p[ip1] = 0;
00099
              for (iq = 0; iq < ctl.nq; iq++)</pre>
00100
              atm2->q[iq][ip1] = 0;
for (ip2 = 0; ip2 < atm2->np; ip2++)
00101
                if (fabs(atm2->time[ip1] - atm2->time[ip2]) < filter_dt) {</pre>
00102
00103
                   atm2->p[ip1] += atm3->p[ip2];
00104
                   for (iq = 0; iq < ctl.nq; iq++)</pre>
00105
                    atm2->q[iq][ip1] += atm3->q[iq][ip2];
00106
                  n++;
00107
              atm2->p[ip1] /= n;
00108
00109
              for (iq = 0; iq < ctl.nq; iq++)</pre>
00110
                atm2->q[iq][ip1] /= n;
00111
00112
            /* Write filtered data... */
00113
           sprintf(filename, "%s.filt", argv[3]);
00114
00115
           write_atm(filename, &ctl, atm2, 0);
00116
00117
00118
         /* Loop over air parcels (reference data)... */
00119
         for (ip2 = 0; ip2 < atm2->np; ip2++) {
00120
00121
            /* Get trajectory length... */
00122
            if (ip2 > 0) {
              geo2cart(0, atm2->lon[ip2 - 1], atm2->lat[ip2 - 1], x1);
geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00123
00124
00125
              lh += DIST(x1, x2);
00126
              lv += fabs(Z(atm2->p[ip2 - 1]) - Z(atm2->p[ip2]));
00127
              lt = fabs(atm2->time[ip2] - atm2->time[0]);
00128
00129
00130
            /* Init... */
           n = 0:
00131
           dh = 0;
00132
           dv = 0;
00133
00134
           for (iq = 0; iq < ctl.nq; iq++)</pre>
00135
             dq[iq] = 0;
00136
            geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00137
            /\star Find corresponding time step (test data)... \star/
00138
           for (ipl = 0; ipl < atml->np; ipl++)
if (fabs(atml->time[ipl] - atm2->time[ip2])
00139
00140
00141
                   < (filter ? filter_dt : 0.1)) {
00142
                /* Calculate deviations... */
geo2cart(0, atm1->lon[ip1], atm1->lat[ip1], x1);
00143
00144
00145
                dh += DIST(x1, x2);
                dv += Z(atm1->p[ip1]) - Z(atm2->p[ip2]);
00146
00147
                for (iq = 0; iq < ctl.nq; iq++)</pre>
00148
                  dq[iq] += atm1->q[iq][ip1] - atm2->q[iq][ip2];
00149
                n++;
00150
00151
           /* Write output... */
00153
            if (n > 0) {
00154
             fprintf(out, "%.2f %.4f %.4f %.4f",
                       atm2->time[ip2], Z(atm2->p[ip2]),
00155
                       atm2->lon[ip2], atm2->lat[ip2]);
00156
              for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl.qnt_format[iq], atm2->q[iq][ip2]);
00157
00158
00159
00160
              fprintf(out, " %.2f %g %g %g %g", lt, lv, lh, dv / n, dh / n);
for (iq = 0; iq < ctl.nq; iq++) {
    fprintf(out, " ");
    fprintf(out, ctl.qnt_format[iq], dq[iq] / n);</pre>
00161
00162
00163
00164
00165
00166
              fprintf(out, "\n");
00167
           }
00168
        }
00169
00170
         /* Close file... */
00171
         fclose(out);
00172
00173
         /* Free... */
00174
         free(atm1);
00175
         free (atm2):
00176
         free(atm3);
00178
         return EXIT_SUCCESS;
00179 }
```

## 5.17 met\_map.c File Reference

Extract global map from meteorological data.

#### **Functions**

• int main (int argc, char \*argv[])

## 5.17.1 Detailed Description

Extract global map from meteorological data.

Definition in file met\_map.c.

#### 5.17.2 Function Documentation

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

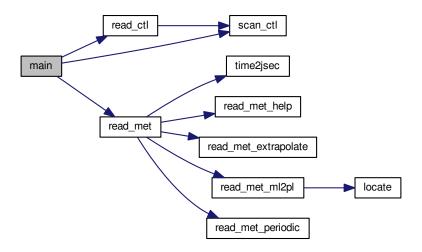
Definition at line 27 of file met\_map.c.

```
00029
                           {
00030
00031
         ctl_t ctl;
00032
00033
         met_t *met;
00034
00035
         FILE *in, *out;
00036
00037
         static double dz, dzmin = 1e10, z, timem[EX][EY], psm[EX][EY], tm[EX][EY],
00038
           um[EX][EY], vm[EX][EY], wm[EX][EY], h2om[EX][EY], o3m[EX][EY];
00039
00040
         static int i, ip, ip2, ix, iy, np[EX][EY];
00041
00042
         /* Allocate... */
00043
         ALLOC(met, met_t, 1);
00044
00045
         /* Check arguments... */
00046
00047
           ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00048
         /* Read control parameters... */
read_ctl(argv[1], argc, argv, &ctl);
z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00049
00050
00051
00052
00053
         /* Loop over files... */
         for (i = 3; i < argc; i++) {</pre>
00054
00055
00056
            /* Read meteorological data... */
00057
           if (!(in = fopen(argv[i], "r")))
00058
              continue;
00059
            else
00060
              fclose(in);
00061
           read_met(&ctl, argv[i], met);
00062
00063
            /* Find nearest pressure level... */
            for (ip2 = 0; ip2 < met->np; ip2++) {
    dz = fabs(Z(met->p[ip2]) - z);
00064
00065
              if (dz < dzmin) {</pre>
00066
00067
                dzmin = dz:
00068
                 ip = ip2;
00069
              }
00070
00071
00072
            /* Average data... */
           for (ix = 0; ix < met->nx; ix++)
  for (iy = 0; iy < met->ny; iy++) {
    timem[ix][iy] += met->time;
    tm[ix][iy] += met->t[ix][iy][ip];
00073
00074
00075
```

```
00077
                um[ix][iy] += met->u[ix][iy][ip];
00078
                vm[ix][iy] += met->v[ix][iy][ip];
                wm[ix][iy] += met->w[ix][iy][ip];
00079
                h2om[ix][iy] += met->h2o[ix][iy][ip];
00080
                o3m[ix][iy] += met->o3[ix][iy][ip];
psm[ix][iy] += met->ps[ix][iy];
00081
00082
00083
               np[ix][iy]++;
00084
00085
00086
00087
         /* Create output file... */
00088
         printf("Write meteorological data file: %s\n", argv[2]);
         if (!(out = fopen(argv[2], "w")))
00089
           ERRMSG("Cannot create file!");
00090
00091
00092
         /* Write header... */
00093
        fprintf(out,
    "# $1
00094
                         = time [s]\n"
                  "# $2
                         = altitude [km]\n"
00096
                         = longitude [deg]\n"
00097
                  "# $4 = latitude [deg]\n"
                  "# $5 = pressure [hPa]\n"
00098
                  "# $6 = temperature [K]\n"
00099
                  "# $7 = zonal wind [m/s]\n"
"# $8 = meridional wind [m/s]\n"
00100
00101
00102
                  "# $9 = vertical wind [hPa/s]\n"
00103
                  "# $10 = H20 volume mixing ratio [1]\n"
                  "# $11 = 03 volume mixing ratio [1]\n"
"# $12 = surface pressure [hPa]\n");
00104
00105
00106
        /* Write data... */
for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
00107
00108
00109
           for (ix = 0; ix < met->nx; ix++)
00110
             00111
00112
00113
00115
00116
00117
00118
           for (ix = 0; ix < met->nx; ix++)
00119
             00120
00121
00122
00123
                         met->lon[ix], met->lat[iy], met->p[ip],
                         mec >lon(|x), mec >lat(|y), mec >p(|x),
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],
psm[ix][iy] / np[ix][iy]);
00124
00125
00126
00127
00128
00129
00130
        /* Close file... */
        fclose(out);
00131
00132
00133
         /* Free... */
00134
        free (met);
00135
00136
         return EXIT_SUCCESS;
00137 }
```

5.18 met\_map.c 139

Here is the call graph for this function:



## 5.18 met\_map.c

```
00001 /*
00002
         This file is part of MPTRAC.
00003
00004
         MPTRAC is free software: you can redistribute it and/or modify
00005
         it under the terms of the GNU General Public License as published by
00006
         the Free Software Foundation, either version 3 of the License, or
00007
         (at your option) any later version.
00008
00009
         MPTRAC is distributed in the hope that it will be useful,
00010
         but WITHOUT ANY WARRANTY; without even the implied warranty of
00011
         MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
         GNU General Public License for more details.
00013
00014
        You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00015
00016
00017
         Copright (C) 2013-2015 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 *in, *out;
00036
         static double dz, dzmin = 1e10, z, timem[EX][EY], psm[EX][EY], tm[EX][EY],
00037
           um[EX][EY], vm[EX][EY], wm[EX][EY], h2om[EX][EY], o3m[EX][EY];
00038
00039
00040
         static int i, ip, ip2, ix, iy, np[EX][EY];
00041
00042
         /* Allocate... */
00043
         ALLOC(met, met_t, 1);
00044
00045
         /* Check arguments... */
00046
         if (argc < 4)
00047
           ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00048
00049
        /* Read control parameters... */
read_ctl(argv[1], argc, argv, &ctl);
z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00050
00051
00052
```

```
00053
        /* Loop over files... */
00054
        for (i = 3; i < argc; i++) {
00055
00056
           /* Read meteorological data... */
           if (!(in = fopen(argv[i], "r")))
00057
00058
             continue;
           else
00059
00060
             fclose(in);
00061
           read_met(&ctl, argv[i], met);
00062
00063
           /* Find nearest pressure level... */
           for (ip2 = 0; ip2 < met->np; ip2++) {
  dz = fabs(Z(met->p[ip2]) - z);
00064
00065
00066
             if (dz < dzmin) {
00067
               dzmin = dz;
00068
               ip = ip2;
00069
             }
00070
           }
00071
00072
           /* Average data... */
00073
           for (ix = 0; ix < met->nx; ix++)
00074
             for (iy = 0; iy < met->ny; iy++) {
00075
               timem[ix][iy] += met->time;
00076
               tm[ix][iy] += met->t[ix][iy][ip];
00077
               um[ix][iy] += met->u[ix][iy][ip];
00078
               vm[ix][iy] += met->v[ix][iy][ip];
00079
                wm[ix][iy] += met->w[ix][iy][ip];
00080
               h2om[ix][iy] += met->h2o[ix][iy][ip];
               o3m[ix][iy] += met->o3[ix][iy][ip];
psm[ix][iy] += met->ps[ix][iy];
00081
00082
00083
               np[ix][iy]++;
00084
00085
00086
00087
         /* Create output file... */
        printf("Write meteorological data file: %s\n", argv[2]);
00088
00089
        if (!(out = fopen(argv[2], "w")))
           ERRMSG("Cannot create file!");
00091
00092
         /* Write header... */
        00093
00094
                  "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n"
00095
00096
00097
                  "# $4
                         = latitude [deg]\n"
00098
                  "# $5 = pressure [hPa]\n"
00099
                  "# $6 = temperature [K]\n"
                  "# $7 = zonal wind [m/s]\n"
"# $8 = meridional wind [m/s]\n"
00100
00101
00102
                  "# $9 = vertical wind [hPa/s] n"
                  "# $10 = H20 volume mixing ratio [1]\n"
00103
                  "# $11 = 03 volume mixing ratio [1]\n"
"# $12 = surface pressure [hPa]\n");
00104
00105
00106
         /* Write data... */
00107
        for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
00108
00110
           for (ix = 0; ix < met->nx; ix++)
             00111
00112
00113
00114
00115
00116
00117
                         h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00118
                        psm[ix][iy] / np[ix][iy]);
           00119
00120
00121
00122
00123
                         met->lon[ix], met->lat[iy], met->p[ip],
                        met >10n(ix), met >1at(iy), met >p(ix) (iy),
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],
psm[ix][iy] / np[ix][iy]);
00124
00125
00126
00127
00128
00129
00130
         /* Close file... */
00131
        fclose(out);
00132
00133
         /* Free... */
        free (met);
00135
00136
        return EXIT_SUCCESS;
00137 }
```

## 5.19 met\_prof.c File Reference

Extract vertical profile from meteorological data.

#### **Functions**

• int main (int argc, char \*argv[])

#### 5.19.1 Detailed Description

Extract vertical profile from meteorological data.

Definition in file met\_prof.c.

## 5.19.2 Function Documentation

#### 5.19.2.1 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 *in, *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],
00048
00049
00050
              w, wm[NZ], h2o, h2om[NZ], o3, o3m[NZ], ps, psm[NZ];
00051
00052
           static int i, iz, np[NZ];
00053
00054
            /* Allocate... */
00055
           ALLOC(met, met_t, 1);
00056
00057
            /* Check arguments... */
00058
00059
              ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00060
00061
           /* Read control parameters... *,
00062
           read_ctl(argv[1], argc, argv, &ctl);
           z0 = scan_ctl(argv[1], argc, argv, "Z0", -1, "0", NULL);

z1 = scan_ctl(argv[1], argc, argv, "Z1", -1, "60", NULL);

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

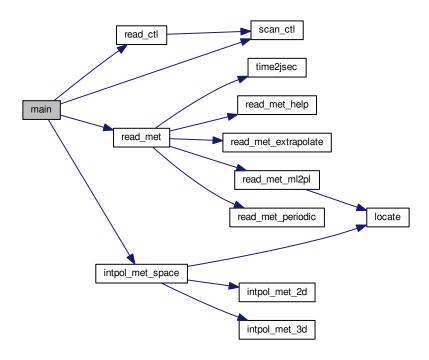
lon0 = scan_ctl(argv[1], argc, argv, "LON0", -1, "0", NULL);

lon1 = scan_ctl(argv[1], argc, argv, "LON0", -1, "0", NULL);
00063
00064
00065
00066
00067
           dlon = scan_ctl(argv[1], argc, argv, "DLON", -1, "1", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "LATO", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "LATO", -1, "0", NULL);
00068
00069
00070
           dlat = scan_ctl(argv[1], argc, argv, "DLAT", -1, "1", NULL);
00071
00072
00073
           /* Loop over input files... */
00074
           for (i = 3; i < argc; i++) {</pre>
00075
              /* Read meteorological data... */
if (!(in = fopen(argv[i], "r")))
00076
00077
00078
                 continue;
00079
              else
08000
                 fclose(in);
00081
              read_met(&ctl, argv[i], met);
00082
00083
               /* Average... */
              for (z = z0; z <= z1; z += dz) {
  iz = (int) ((z - z0) / dz);
  if (iz < 0 || iz > NZ)
00084
00085
00086
00087
                    ERRMSG("Too many altitudes!");
```

```
00088
             for (lon = lon0; lon <= lon1; lon += dlon)</pre>
               for (lat = lat0; lat <= lat1; lat += dlat) {
  intpol_met_space(met, P(z), lon, lat, &ps,</pre>
00089
00090
                  &t, &u, &v, &w, &h2o, &o3); if (gsl_finite(t) && gsl_finite(u)
00091
00092
00093
                       && qsl_finite(v) && qsl_finite(w)) {
                    timem[iz] += met->time;
00095
                     lonm[iz] += lon;
                    latm[iz] += lat;
tm[iz] += t;
00096
00097
                    um[iz] += u;
00098
00099
                    vm[iz] += v;
                     wm[iz] += w;
00100
00101
                    h2om[iz] += h2o;
00102
                    o3m[iz] += o3;
                     psm[iz] += ps;
00103
00104
                     np[iz]++;
00105
                  }
00106
00107
           }
        }
00108
00109
        /* Normalize... */
for (z = z0; z <= z1; z += dz) {
  iz = (int) ((z - z0) / dz);
  if (np[iz] > 0) {
00110
00111
00112
00113
00114
             timem[iz] /= np[iz];
             lonm[iz] /= np[iz];
latm[iz] /= np[iz];
00115
00116
             tm[iz] /= np[iz];
um[iz] /= np[iz];
00117
00118
00119
             vm[iz] /= np[iz];
00120
              wm[iz] /= np[iz];
00121
             h2om[iz] /= np[iz];
             o3m[iz] /= np[iz];
psm[iz] /= np[iz];
00122
00123
00124
           } else {
             timem[iz] = GSL_NAN;
00126
              lonm[iz] = GSL_NAN;
             latm[iz] = GSL_NAN;
00127
             tm[iz] = GSL_NAN;
um[iz] = GSL_NAN;
vm[iz] = GSL_NAN;
00128
00129
00130
              wm[iz] = GSL_NAN;
00131
00132
             h2om[iz] = GSL_NAN;
00133
              o3m[iz] = GSL_NAN;
             psm[iz] = GSL_NAN;
00134
00135
        }
00136
00137
00138
         /* Create output file... */
00139
         printf("Write meteorological data file: %s\n", argv[2]);
00140
         if (!(out = fopen(argv[2], "w")))
00141
           ERRMSG("Cannot create file!");
00142
00143
         /* Write header... */
        fprintf(out,
00145
                  "# $1
                          = time [s]\n"
                  "# $2 = altitude [km]\n"
00146
                  "# $3 = longitude [deg]\n"
"# $4 = latitude [deg]\n"
"# $5 = pressure [hPa]\n"
00147
00148
00149
00150
                  "# $6 = temperature [K]\n"
00151
                  "# $7 = zonal wind [m/s]\n"
00152
                  "# $8 = meridional wind [m/s]\n"
                  "# $9 = vertical wind [hPa/s]\n"
00153
                  "# $10 = H20 volume mixing ratio [1]\n"
00154
                  "# $11 = 03 volume mixing ratio [1]\n"
"# $12 = surface pressure [hPa]\n\n");
00155
00156
00157
        00158
00159
00160
00161
00162
00163
00164
00165
        /* Close file... */
00166
        fclose(out):
00167
00168
00169
         /* Free... */
00170
        free (met);
00171
00172
        return EXIT_SUCCESS;
00173 }
```

5.20 met prof.c 143

Here is the call graph for this function:



# 5.20 met\_prof.c

```
00001 /*
00002
         This file is part of MPTRAC.
00003
00004
         MPTRAC is free software: you can redistribute it and/or modify
00005
         it under the terms of the GNU General Public License as published by
         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
         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
         Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00025 #include "libtrac.h"
00026
00027 /* -----
00028
          Dimensions...
00029
00030
00031 /\star Maximum number of altitudes. \star/
00032 #define NZ 1000 \,
00033
00034 /* --
00035
          Main...
00036
00037
00038 int main(
00039
         int argc,
00040
         char *argv[]) {
00041
00042
         ctl_t ctl;
00043
```

```
00044
          met_t *met;
00045
00046
           FILE *in, *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];
00048
00049
00051
00052
           static int i, iz, np[NZ];
00053
00054
            /* Allocate... */
00055
           ALLOC(met, met_t, 1);
00056
00057
            /* Check arguments... */
00058
           if (argc < 4)
00059
              ERRMSG("Give parameters: <ctl>   <met0> [ <met1> ... ]");
00060
00061
           /* Read control parameters... */
           read_ctl(argv[1], argc, argv, &ctl);
00062
          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, "LONO", -1, "0", NULL);
dlon = scan_ctl(argv[1], argc, argv, "DLON", -1, "1", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "LATO", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "LAT1", -1, "0", NULL);
dlat = scan_ctl(argv[1], argc, argv, "DLAT", -1, "1", NULL);
00063
00064
00065
00066
00067
00068
00069
00070
00071
00072
00073
           /* Loop over input files... */
00074
           for (i = 3; i < argc; i++) {
00075
00076
               /* Read meteorological data... */
00077
              if (!(in = fopen(argv[i], "r")))
00078
                 continue;
00079
              else
08000
                fclose(in);
              read_met(&ctl, argv[i], met);
00082
00083
               /* Average... */
              for (z = z0; z <= z1; z += dz) {
  iz = (int) ((z - z0) / dz);
00084
00085
                 <u>if</u> (iz < 0 || iz > NZ)
00086
00087
                    ERRMSG("Too many altitudes!");
                  for (lon = lon0; lon <= lon1; lon += dlon)</pre>
00088
00089
                    for (lat = lat0; lat <= lat1; lat += dlat) {</pre>
00090
                       intpol_met_space(met, P(z), lon, lat, &ps,
                       &t, &u, &v, &w, &h2o, &o3);
if (gsl_finite(t) && gsl_finite(u)
00091
00092
00093
                            && gsl_finite(v) && gsl_finite(w)) {
00094
                          timem[iz] += met->time;
00095
                          lonm[iz] += lon;
                          latm[iz] += lat;
00096
00097
                          tm[iz] += t;
                          um[iz] += u;
00098
00099
                          vm[iz] += v;
                          wm[iz] += w;
00100
00101
                          h2om[iz] += h2o;
00102
                          o3m[iz] += o3;
00103
                          psm[iz] += ps;
00104
                          np[iz]++;
00105
00106
                    }
00107
             }
00108
00109
00110
           /* Normalize... */
           for (z = z0; z <= z1; z += dz) {
  iz = (int) ((z - z0) / dz);
00111
00112
              if (np[iz] > 0) {
00113
00114
                timem[iz] /= np[iz];
                 lonm[iz] /= np[iz];
00115
                 latm[iz] /= np[iz];
00116
                 tm[iz] /= np[iz];
um[iz] /= np[iz];
vm[iz] /= np[iz];
00117
00118
00119
00120
                 wm[iz] /= np[iz];
00121
                 h2om[iz] /= np[iz];
                o3m[iz] /= np[iz];
psm[iz] /= np[iz];
00122
00123
00124
              } else {
00125
                 timem[iz] = GSL_NAN;
                 lonm[iz] = GSL_NAN;
latm[iz] = GSL_NAN;
00126
00127
                tm[iz] = GSL_NAN;
um[iz] = GSL_NAN;
00128
00129
                 vm[iz] = GSL_NAN;
00130
```

```
wm[iz] = GSL_NAN;
             h2om[iz] = GSL_NAN;
o3m[iz] = GSL_NAN;
00132
00133
             psm[iz] = GSL_NAN;
00134
00135
00136
00137
00138
        /* Create output file... */
00139
        printf("Write meteorological data file: %s\n", argv[2]);
00140
         if (!(out = fopen(argv[2], "w")))
           ERRMSG("Cannot create file!");
00141
00142
00143
         /* Write header... */
00144
                         = time [s]\n"
00145
                  "# $1
                  "# $1 = Line [5]\n"
"# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n"
"# $4 = latitude [deg]\n"
"# $5 = pressure [hPa]\n"
00146
00147
00148
                 "# $6 = temperature [K]\n"
"# $7 = zonal wind [m/s]\n"
00150
00151
                  "# $8 = meridional wind [m/s]\n"
00152
                  "# $9 = vertical wind [hPa/s]\n"
00153
                  "# $10 = H20 volume mixing ratio [1]\n"
00154
                  "# $11 = 03 volume mixing ratio [1]\n"
"# $12 = surface pressure [hPa]\n\n");
00155
00156
00157
        /* Write data... */
for (z = z0; z <= z1; z += dz) {
00158
00159
         iz = (int) ((z - z0) / dz);
00160
          00161
00162
00163
00164
00165
        /* Close file... */
00166
00167
        fclose(out);
00168
         /* Free... */
00170
        free (met);
00171
00172
         return EXIT_SUCCESS;
00173 }
```

## 5.21 met\_sample.c File Reference

Sample meteorological data at given geolocations.

# Functions

• int main (int argc, char \*argv[])

## 5.21.1 Detailed Description

Sample meteorological data at given geolocations.

Definition in file met\_sample.c.

### 5.21.2 Function Documentation

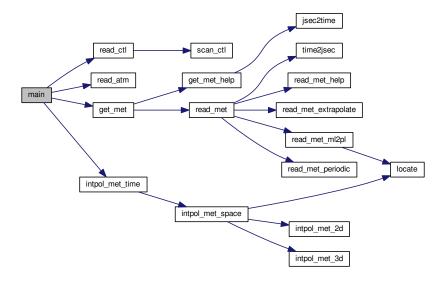
## 5.21.2.1 int main ( int argc, char \* argv[] )

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

```
00033
                       {
00034
00035
        ctl_t ctl;
00036
00037
        atm t *atm;
00038
        met_t *met0, *met1;
00040
00041
       FILE *out;
00042
00043
        double t, u, v, w, h2o, o3;
00044
00045
        int ip;
00046
00047
        /\star Check arguments... \star/
        if (argc < 4)
    ERRMSG("Give parameters: <ctl> <metbase> <atm_in> <sample.tab>");
00048
00049
00050
00051
        /* Allocate... */
00052
        ALLOC(atm, atm_t, 1);
        ALLOC(met0, met_t, 1);
ALLOC(met1, met_t, 1);
00053
00054
00055
00056
        /* Read control parameters... */
00057
        read_ctl(argv[1], argc, argv, &ctl);
00058
00059
        /* Read atmospheric data... */
00060
        read_atm(argv[3], &ctl, atm);
00061
00062
        /* Create output file... */
        printf("Write meteorological data file: %s\n", argv[4]);
if (!(out = fopen(argv[4], "w")))
00063
00064
00065
          ERRMSG("Cannot create file!");
00066
00067
        /* Write header... */
        fprintf(out,
    "# $1
00068
00069
                       = time [s]\n"
00070
                 "# $2 = altitude [km]\n"
00071
                 "# $3 = longitude [deg] \n"
00072
                 "# $4 = latitude [deg]\n"
                 "# $5 = pressure [hPa]\n"
"# $6 = temperature [K]\n"
"# $7 = zonal wind [m/s]\n"
00073
00074
00075
                 "# $8 = meridional wind [m/s]\n'
00076
00077
                 "# $9 = vertical wind [hPa/s]\n"
                 "# $10 = H2O volume mixing ratio [1]\n"
00078
00079
                 "# $11 = 03 volume mixing ratio [1]\n\n");
08000
       /* Loop over air parcels... */
for (ip = 0; ip < atm->np; ip++) {
00081
00082
00083
00084
           /* Get meteorological data... */
00085
          get_met(&ctl, argv[2], atm->time[ip], met0, met1);
00086
          /* Interpolate meteorological data... */
00087
           intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00088
      lon[ip],
00089
                            atm->lat[ip], NULL, &t, &u, &v, &w, &h2o, &o3);
00090
          00091
00092
00093
00094
00095
00096
00097
        /* Close file... */
00098
        fclose(out);
00099
00100
        /* Free... */
00101
        free(atm);
00102
        free (met0);
00103
        free (met1);
00104
00105
        return EXIT_SUCCESS;
00106 }
```

5.22 met sample.c 147

Here is the call graph for this function:



## 5.22 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
the Free Software Foundation, either version 3 of the License, or
00005
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
         Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00017
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /*
00028
          Main...
00029
00030
00031 int main(
00032
         int argc,
00033
         char *argv[]) {
00034
00035
         ctl_t ctl;
00036
00037
         atm t *atm;
00038
00039
         met_t *met0, *met1;
00040
00041
        FILE *out;
00042
00043
         double t, u, v, w, h2o, o3;
00044
00045
         int ip;
00046
00047
         /* Check arguments... */
00048
         if (argc < 4)
           ERRMSG("Give parameters: <ctl> <metbase> <atm_in> <sample.tab>");
00049
00050
00051
         /* Allocate... */
```

```
00052
       ALLOC(atm, atm_t, 1);
       ALLOC (met1, met_t, 1);
ALLOC (met1, met_t, 1);
00053
00054
00055
00056
       /* Read control parameters... */
00057
       read_ctl(argv[1], argc, argv, &ctl);
00058
00059
        /* Read atmospheric data... */
00060
       read_atm(argv[3], &ctl, atm);
00061
00062
       /* Create output file... */
       printf("Write meteorological data file: %s\n", argv[4]);
if (!(out = fopen(argv[4], "w")))
00063
00064
         ERRMSG("Cannot create file!");
00065
00066
00067
        /* Write header... */
       00068
00069
00070
                "# $2 = altitude [km] \n"
00071
                "# $3 = longitude [deg]\n"
00072
                "# $4 = latitude [deg]\n"
                "# $5 = pressure [hPa]\n"
00073
                "# $6 = temperature [K]\n"
00074
               "# $7 = zonal wind [m/s]\n"
"# $8 = meridional wind [m/s]\n'
00075
00076
00077
                "# $9 = vertical wind [hPa/s]\n"
00078
                "# $10 = H20 volume mixing ratio [1]\n"
                "# $11 = 03 volume mixing ratio [1]\n\n");
00079
08000
00081
       /* Loop over air parcels... */
00082
       for (ip = 0; ip < atm->np; ip++) {
00083
00084
          /* Get meteorological data... */
00085
         get_met(&ctl, argv[2], atm->time[ip], met0, met1);
00086
         /* Interpolate meteorological data... */
00087
00088
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
     lon[ip],
00089
                          atm->lat[ip], NULL, &t, &u, &v, &w, &h2o, &o3);
00090
         00091
00092
00093
00094
00095
00096
00097
       /* Close file... */
00098
       fclose(out);
00099
       /* Free... */
00100
00101
       free(atm);
       free(met0);
00102
00103
       free (met1);
00104
       return EXIT_SUCCESS;
00105
00106 }
```

## 5.23 met\_zm.c File Reference

Extract zonal mean from meteorological data.

### **Functions**

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

### 5.23.1 Detailed Description

Extract zonal mean from meteorological data.

Definition in file met\_zm.c.

#### 5.23.2 Function Documentation

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

Definition at line 27 of file met zm.c.

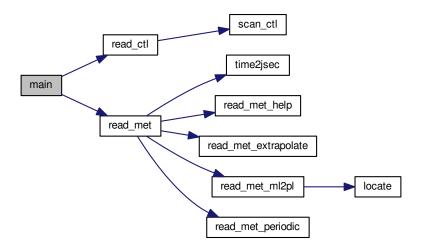
```
00029
00030
00031
                ctl_t ctl;
00032
00033
               met t *met;
00034
                FILE *in, *out;
00036
00037
                static double timem[EP][EY], psm[EP][EY], tm[EP][EY], um[EP][EY],
                   vm(EP][EY], vhm[EP][EY], wm[EP][EY], h2om[EP][EY], o3m[EP][EY],
psm2[EP][EY], tm2[EP][EY], um2[EP][EY], vm2[EP][EY], vhm2[EP][EY],
wm2[EP][EY], h2om2[EP][EY], o3m2[EP][EY];
00038
00039
00040
00041
00042
                static int i, ip, ix, iy, np[EP][EY];
00043
00044
                 /* Allocate... */
00045
                ALLOC(met, met_t, 1);
00046
00047
                /* Check arguments... */
00048
                if (argc < 4)
00049
                    ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00050
00051
                /* Read control parameters... */
00052
                read_ctl(argv[1], argc, argv, &ctl);
00053
00054
                 /* Loop over files... */
00055
                for (i = 3; i < argc; i++) {</pre>
00056
                    /* Read meteorological data... */
if (!(in = fopen(argv[i], "r")))
00057
00058
00059
                        continue;
00060
00061
                        fclose(in);
00062
                    read_met(&ctl, argv[i], met);
00063
00064
                     /* Average data... */
                    for (ix = 0; ix < met->nx; ix++)
00065
                        for (iy = 0; iy < met->ny; iy++)
00066
00067
                             for (ip = 0; ip < met->np; ip++) {
00068
                                 timem[ip][iy] += met->time;
                                 tm[ip][iy] += met->t[ix][iy][ip];
um[ip][iy] += met->u[ix][iy][ip];
00069
00070
                                 vm[ip][iy] += met->v[ix][iy][ip];
00071
00072
                                 vhm[ip][iy] += sqrt(gsl_pow_2(met->u[ix][iy][ip])
00073
                                                                            + gsl_pow_2 (met->v[ix][iy][ip]));
00074
                                 wm[ip][iy] += met->w[ix][iy][ip];
                                 h2om[ip][iy] += met->h2o[ix][iy][ip];
00075
                                 o3m[ip][iy] += met->o3[ix][iy][ip];
psm[ip][iy] += met->ps[ix][iy];
00076
00077
00078
                                 tm2[ip][iy] += gsl_pow_2(met->t[ix][iy][ip]);
00079
                                 um2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip]);
00080
                                 vm2[ip][iy] += gsl_pow_2(met->v[ix][iy][ip]);
                                 \label{limits} $$ vhm2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip])$
00081
                                    + gsl_pow_2 (met->v[ix][iy][ip]);
00082
                                 wm2[ip][iy] += gsl_pow_2(met->w[ix][iy][ip]);
00083
                                 h2om2[ip][iy] += gsl_pow_2(met->h2o[ix][iy][ip]);
00084
                                 parallels();

00085
00086
00087
                                 np[ip][iy]++;
00088
                            }
00089
00090
00091
                 /* Create output file... */
00092
                printf("Write meteorological data file: %s\n", argv[2]);
                if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");
00093
00094
00095
00096
                /* Write header... */
00097
                fprintf(out,
                                              = time [s] \n"
00098
                                 "# $1
                                              = altitude [km] n"
00099
                                 "# $2
                                 "# $3 = latitude [deg] \n"
00100
                                 "# $4 = temperature mean [K]\n"
00101
00102
                                 "# $5 = temperature standard deviation [K]\n"
00103
                                 "# $6 = zonal wind mean [m/s]\n"
                                 "# \$7 = zonal wind standard deviation [m/s]\n"
00104
```

```
"# $8 = meridional wind mean [m/s]\n"
00106
                "# $9 = meridional wind standard deviation [m/s]\n"
                "# $10 = horizontal wind mean [m/s]\n"
00107
                "# $11 = horizontal wind standard deviation [m/s]\n"
00108
                "# $12 = vertical wind mean [hPa/s]\n"
"# $13 = vertical wind standard deviation [hPa/s]\n"
00109
00110
                "# $14 = H20 vmr mean [1]\n"
00111
00112
                "# $15 = H20 \text{ vmr standard deviation } [1] \n"
00113
                "# $16 = 03 \text{ vmr mean } [1] \n"
                "# $17 = 03 vmr standard deviation [1]\n"
00114
                "# $18 = surface pressure mean [hPa]\n"
00115
                "# $19 = surface pressure standard deviation [hPa] \n");
00116
00117
        /* Write data... */
00118
        00119
00120
00121
00122
                    00124
00125
00126
00127
00128
00129
00130
00131
                    vm[ip][iy] / np[ip][iy],
                    00132
00133
00134
00135
00136
00137
                    sqrt(wm2[ip][iy] / np[ip][iy] -
    gsl_pow_2(wm[ip][iy] / np[ip][iy]),
h2om[ip][iy] / np[ip][iy],
sqrt(h2om2[ip][iy] / np[ip][iy] -
    gsl_pow_2(h2om[ip][iy] / np[ip][iy])),
00138
00139
00140
00141
                    o3m[ip][iy] / np[ip][iy],
sqrt(o3m2[ip][iy] / np[ip][iy] -
00143
00144
                         gsl_pow_2(o3m[ip][iy] / np[ip][iy])),
00145
                    00146
00147
00148
00149
00150
00151
        /* Close file... */
00152
       fclose(out);
00153
00154
        /* Free... */
00155
        free (met);
00156
00157
        return EXIT_SUCCESS;
00158 }
```

5.24 met zm.c 151

Here is the call graph for this function:



# 5.24 met\_zm.c

```
00001 /*
00002
         This file is part of MPTRAC.
00003
00004
         MPTRAC is free software: you can redistribute it and/or modify
00005
         it under the terms of the GNU General Public License as published by
00006
         the Free Software Foundation, either version 3 of the License, or
00007
         (at your option) any later version.
00008
00009
         MPTRAC is distributed in the hope that it will be useful,
00010
         but WITHOUT ANY WARRANTY; without even the implied warranty of
00011
         MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
         GNU General Public License for more details.
00013
00014
         You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00015
00016
00017
         Copright (C) 2013-2015 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 *in, *out;
00036
         static double timem[EP][EY], psm[EP][EY], tm[EP][EY], um[EP][EY],
00037
           vm[EP][EY], vhm[EP][EY], wm[EP][EY], h2om[EP][EY], o3m[EP][EY],
psm2[EP][EY], tm2[EP][EY], um2[EP][EY], vm2[EP][EY], vhm2[EP][EY],
wm2[EP][EY], h2om2[EP][EY], o3m2[EP][EY];
00038
00039
00040
00041
00042
         static int i, ip, ix, iy, np[EP][EY];
00043
00044
         /* Allocate... */
00045
         ALLOC(met, met_t, 1);
00046
00047
         /* Check arguments... */
00048
         if (argc < 4)
00049
           ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00050
00051
         /* Read control parameters... */
00052
         read_ctl(argv[1], argc, argv, &ctl);
```

```
00053
00054
              /* Loop over files... */
00055
              for (i = 3; i < argc; i++) {</pre>
00056
00057
                  /* Read meteorological data...
                  if (!(in = fopen(argv[i], "r")))
00058
00059
                     continue;
00060
                  else
00061
                     fclose(in);
00062
                  read_met(&ctl, argv[i], met);
00063
00064
                 /* Average data... */
for (ix = 0; ix < met->nx; ix++)
00065
00066
                      for (iy = 0; iy < met->ny; iy++)
00067
                         for (ip = 0; ip < met->np; ip++) {
00068
                             timem[ip][iy] += met->time;
                             tm[ip][iy] += met->t[ix][iy][ip];
um[ip][iy] += met->u[ix][iy][ip];
vm[ip][iy] += met->v[ix][iy][ip];
00069
00070
00072
                             vhm[ip][iy] += sqrt(gsl_pow_2(met->u[ix][iy][ip])
00073
                                                                  + gsl_pow_2 (met->v[ix][iy][ip]));
                             wm[ip][iy] += met->w[ix][iy][ip];
00074
00075
                             h2om[ip][iy] += met->h2o[ix][iy][ip];
00076
                             o3m[ip][iy] += met->o3[ix][iy][ip];
                             OSM[tp][ty] += met->ps[ix][ty][tp],
psm[ip][iy] += met->ps[ix][iy];
tm2[ip][iy] += gsl_pow_2(met->t[ix][iy][ip]);
00077
00078
00079
                             um2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip]);
00080
                             vm2[ip][iy] += gsl_pow_2(met->v[ix][iy][ip]);
                             vhm2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip])
00081
                             + gsl_pow_2 (met->v[ix][iy][ip]);
wm2[ip][iy] += gsl_pow_2 (met->w[ix][iy][ip]);
00082
00083
                             Mind if it is a state of the state of t
00084
00085
00086
                             psm2[ip][iy] += gsl_pow_2(met->ps[ix][iy]);
00087
                             np[ip][iy]++;
00088
00089
              }
00090
00091
               /* Create output file... */
00092
              printf("Write meteorological data file: %s\n", argv[2]);
00093
              if (!(out = fopen(argv[2], "w")))
                 ERRMSG("Cannot create file!");
00094
00095
00096
              /* Write header... */
00097
              fprintf(out,
00098
                             "# $1
                                         = time [s]\n"
                             "# $2 = altitude [km]\n"
"# $3 = latitude [deg]\n"
00099
00100
                             "# $4 = temperature mean [K]\n"
00101
00102
                             "# $5 = temperature standard deviation [K]\n"
00103
                             "# $6 = zonal wind mean [m/s]\n"
00104
                             "# $7 = zonal wind standard deviation [m/s]\n"
00105
                             "# $8 = meridional wind mean [m/s]\n"
00106
                             "# $9 = meridional wind standard deviation [m/s]\n"
                             "# $10 = horizontal wind mean [m/s]\n"
00107
                             "# $11 = horizontal wind standard deviation [m/s]\n"
00108
                             "# $12 = vertical wind mean [hPa/s]\n"
00110
                             "# $13 = vertical wind standard deviation [hPa/s]\n"
00111
                             "# $14 = H20 vmr mean [1]\n"
                             "# $15 = H20 \text{ vmr standard deviation } [1] \n"
00112
                             "# $16 = 03 vmr mean [1] n"
00113
                             "# $17 = 03 vmr standard deviation [1]\n"
00114
00115
                             "# $18 = surface pressure mean [hPa]\n'
                             "# $19 = surface pressure standard deviation [hPa]\n");
00116
00117
              /* Write data... */
00118
             00119
00120
00121
00122
                                    " %g %g %g %g %g %g %g %g,\n",
timem[ip][iy] / np[ip][iy], Z(met->p[ip]), met->lat[iy],
00123
00124
                                    tm[ip][iy] / np[ip][iy],
sqrt(tm2[ip][iy] / np[ip][iy] -
00125
00126
                                    gsl_pow_2(tm[ip][iy] / np[ip][iy])),
um[ip][iy] / np[ip][iy],
00127
00128
                                    00129
00130
                                   00131
00132
00133
00134
00135
00136
                                             gsl_pow_2(vhm[ip][iy] / np[ip][iy])),
                                    00137
00138
00139
```

```
h2om[ip][iy] / np[ip][iy],
                     00142
                     o3m[ip][iy] / np[ip][iy],
sqrt(o3m2[ip][iy] / np[ip][iy] -
00143
00144
                     gsl_pow_2(o3m[ip][iy] / np[ip][iy])),
psm[ip][iy] / np[ip][iy],
sqrt(psm2[ip][iy] / np[ip][iy] -
00145
00146
00147
00148
                          gsl_pow_2(psm[ip][iy] / np[ip][iy])));
00149
00150
        /* Close file... */
00151
00152
        fclose(out);
00153
00154
        /* Free... */
00155
        free(met);
00156
        return EXIT_SUCCESS;
00157
00158 }
```

## 5.25 smago.c File Reference

Estimate horizontal diffusivity based on Smagorinsky theory.

#### **Functions**

• int main (int argc, char \*argv[])

### 5.25.1 Detailed Description

Estimate horizontal diffusivity based on Smagorinsky theory.

Definition in file smago.c.

### 5.25.2 Function Documentation

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

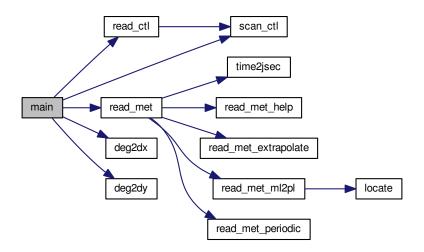
Definition at line 8 of file smago.c.

```
00010
00011
00012
        ctl_t ctl;
00013
00014
        met t *met;
00015
00016
00017
        static double dz, dzmin = 1e10, z, t, s, 1s2, k[EX][EY], c = 0.15;
00018
00019
00020
        static int ip, ip2, ix, iv;
00021
00022
         /* Allocate... */
00023
        ALLOC(met, met_t, 1);
00024
        /* Check arguments... */
if (argc < 4)</pre>
00025
00026
          ERRMSG("Give parameters: <ctl> <map.tab> <met>");
00027
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);
  if (dz < dzmin) {</pre>
00037
00038
00039
            dzmin = dz;
00040
00041
            ip = ip2;
00042
00043
00044
        /* Write info... */
printf("Analyze %g hPa...\n", met->p[ip]);
00045
00046
00047
00048
         /* Calculate horizontal diffusion coefficients... */
        for (ix = 1; ix < met->nx - 1; ix++)
for (iy = 1; iy < met->ny - 1; iy++) {
00049
00050
             t = 0.5 * ((met->u[ix + 1][iy][ip] - met->u[ix - 1][iy][ip])
00051
                         / (1000. *
00052
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])));
             00056
00057
00058
00059
                         / (1000. *
                            deg2dx(met->lon[ix + 1] - met->lon[ix - 1],
00060
00061
                                   met->lat[iy])));
            ls2 = gsl_pow_2(c * 500. * deg2dy(met->lat[iy + 1] - met->lat[iy - 1]));
if (fabs(met->lat[iy]) > 80)
00062
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]);
if (!(out = fopen(argv[2], "w")))
00068
00069
00070
00071
           ERRMSG("Cannot create file!");
00072
00073
         /* Write header... */
        00074
00075
                 "# $2 = latitude [deg]\n"
00076
                 "# $3 = zonal wind [m/s]\n"
00077
00078
                 "# $4 = meridional wind [m/s]\n"
00079
                 "# $5 = horizontal diffusivity [m^2/s]\n");
08000
00081
        /\star Write data... \star/
        for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
  for (ix = 0; ix < met->nx; ix++)
00082
00083
00084
00085
             if (met \rightarrow lon[ix] >= 180)
               00086
00087
00088
00089
           for (ix = 0; ix < met->nx; ix++)
            if (met->lon[ix] <= 180)</pre>
00090
00091
               fprintf(out, "%g %g %g %g %g\n",
00092
                        met->lon[ix], met->lat[iy],
00093
                        met \rightarrow u[ix][iy][ip], met \rightarrow v[ix][iy][ip], k[ix][iy]);
00094
00095
00096
         /* Close file... */
00097
        fclose(out);
00098
         /* Free... */
00099
00100
        free (met);
00101
00102
        return EXIT_SUCCESS;
00103 }
```

5.26 smago.c 155

Here is the call graph for this function:



## 5.26 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 }
```

## 5.27 split.c File Reference

Split air parcels into a larger number of parcels.

#### **Functions**

• int main (int argc, char \*argv[])

### 5.27.1 Detailed Description

Split air parcels into a larger number of parcels.

Definition in file split.c.

#### 5.27.2 Function Documentation

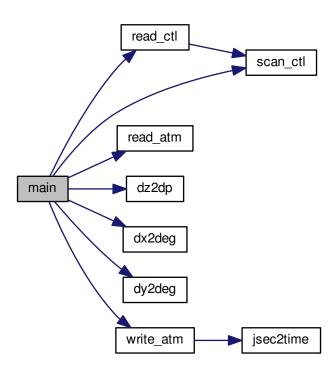
#### 5.27.2.1 int main ( int argc, char \* argv[])

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:



5.28 split.c 159

### 5.28 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
          Copright (C) 2013-2015 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 - > np] = 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 }
```

## 5.29 time2jsec.c File Reference

Convert date to Julian seconds.

#### **Functions**

• int main (int argc, char \*argv[])

### 5.29.1 Detailed Description

Convert date to Julian seconds.

Definition in file time2jsec.c.

5.30 time2jsec.c 161

### 5.29.2 Function Documentation

### 5.29.2.1 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:



# 5.30 time2jsec.c

```
00001 /*
00002
        This file is part of MPTRAC.
00003
        MPTRAC is free software: you can redistribute it and/or modify
00004
         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
        Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028
      int argc,
00029
        char *argv[]) {
```

```
00030
00031
        double jsec, remain;
00032
00033
       int day, hour, min, mon, sec, year;
00034
00035
        /* Check arguments... */
00036
       if (argc < 8)
00037
          ERRMSG("Give parameters: <year> <mon> <day> <hour> <min> <sec> <remain>");
00038
00039
        /* Read arguments... */
00040
       year = atoi(argv[1]);
00041
        mon = atoi(argv[2]);
00042
        day = atoi(argv[3]);
00043
        hour = atoi(argv[4]);
00044
       min = atoi(argv[5]);
       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 }
```

### 5.31 trac.c File Reference

Lagrangian particle dispersion model.

#### **Functions**

void init\_simtime (ctl\_t \*ctl, atm\_t \*atm)

Set simulation time interval.

• 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[])

## 5.31.1 Detailed Description

Lagrangian particle dispersion model.

Definition in file trac.c.

5.31 trac.c File Reference 163

#### 5.31.2 Function Documentation

#### 5.31.2.1 void init\_simtime ( ctl\_t \* ctl, atm\_t \* atm )

Set simulation time interval.

Definition at line 398 of file trac.c.

```
00400
00401
         /\star Set inital and final time... \star/
00403
        if (ctl->direction == 1)
00404
          if (ctl->t_start < -1e99)
00405
             ctl->t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
        if (ctl->t_stop < -le99)
  ctl->t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
} else if (ctl->direction == -1) {
00406
00407
00408
00409
         if (ctl->t_stop < -1e99)</pre>
00410
             ctl->t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00411
           if (ctl->t_start < -1e99)</pre>
             ctl->t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00412
00413
00414
         /* Check time... */
00416
         if (ctl->direction * (ctl->t_stop - ctl->t_start) <= 0)</pre>
00417
           ERRMSG("Nothing to do!");
00418 }
```

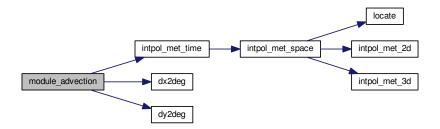
5.31.2.2 void module\_advection ( met\_t \* met0, met\_t \* met1, atm\_t \* atm, int ip, double dt )

Calculate advection of air parcels.

Definition at line 422 of file trac.c.

```
{
00428
00429
          double v[3], xm[3];
00430
00431
           /* Interpolate meteorological data... */
           intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00432
00433
                                 atm->lon[ip], atm->lat[ip], NULL, NULL,
00434
                                  &v[0], &v[1], &v[2], NULL, NULL);
00435
          /* Get position of the mid point... */ xm[0] = atm->lon[ip] + dx2deg(0.5 * dt * v[0] / 1000., atm->lat[ip]); <math>xm[1] = atm->lat[ip] + dy2deg(0.5 * dt * v[1] / 1000.);
00436
00437
00438
00439
           xm[2] = atm - p[ip] + 0.5 * dt * v[2];
00440
00441
           /\star Interpolate meteorological data for mid point... \star/
            \begin{array}{c} intpol\_met\_time (met0, met1, atm->time[ip] + 0.5 * dt, \\ xm[2], xm[0], xm[1], NULL, NULL, \\ &v[0], &v[1], &v[2], NULL, NULL); \end{array} 
00442
00443
00444
00445
00446
          /* Save new position... */
00447
           atm->time[ip] += dt;
          atm->lan[ip] += dx2deg(dt * v[0] / 1000., xm[1]);
atm->lat[ip] += dy2deg(dt * v[1] / 1000.);
00448
00449
00450
          atm->p[ip] += dt * v[2];
00451 }
```

Here is the call graph for this function:



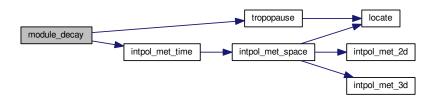
5.31.2.3 void module\_decay ( ctl\_t \* ctl, met\_t \* met0, met\_t \* met1, atm\_t \* atm, int ip, double dt )

Calculate exponential decay of particle mass.

Definition at line 455 of file trac.c.

```
00461
                    {
00462
00463
        double ps, pt, tdec;
00464
00465
        /* Check lifetime values... */
qnt_m < 0)
00467
00466
         if ((ctl->tdec_trop <= 0 && ctl->tdec_strat <= 0) || ctl->
          return;
00468
00469
        /* Set constant lifetime... */
00470
        if (ctl->tdec_trop == ctl->tdec_strat)
00471
          tdec = ctl->tdec_trop;
00472
00473
        /* Set altitude-dependent lifetime... */
00474
        else {
00475
00476
           /* Get surface pressure... */
00477
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00478
                           atm->lon[ip], atm->lat[ip], &ps, NULL,
00479
                           NULL, NULL, NULL, NULL, NULL);
00480
00481
          /* Get tropopause pressure... */
00482
          pt = tropopause(atm->time[ip], atm->lat[ip]);
00483
00484
          /\star Set lifetime... \star/
00485
          if (atm->p[ip] \le pt)
00486
            tdec = ctl->tdec_strat;
00487
          else
00488
            tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
     p[ip]);
00489
00490
        /* Calculate exponential decay... */
atm->q[ctl->qnt_m][ip] *= exp(-dt / tdec);
00491
00492
00493 }
```

Here is the call graph for this function:



5.31.2.4 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.

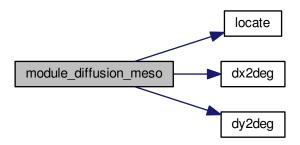
Definition at line 497 of file trac.c.

```
{
00505
00506
         double r, rs, u[16], v[16], w[16], usig, vsig, wsig;
00507
00508
         int ix, iv, iz;
00509
00510
         /* Calculate mesoscale velocity fluctuations... */
00511
         if (ctl->turb_meso > 0) {
00512
00513
            /* Get indices... */
           ix = locate(met0->lon, met0->nx, atm->lon[ip]);
00514
00515
            iy = locate(met0->lat, met0->ny, atm->lat[ip]);
00516
           iz = locate(met0->p, met0->np, atm->p[ip]);
00517
00518
            /* Collect local wind data... */
           u[0] = met0->u[ix][iy][iz];
u[1] = met0->u[ix + 1][iy][iz];
00519
00520
           u[2] = met0 >u[ix | 1][iy][12],
u[2] = met0 >u[ix][iy + 1][iz];
u[3] = met0 >u[ix + 1][iy + 1][iz];
00521
           u[4] = met0 -> u[ix][iy][iz + 1];
00523
00524
            u[5] = met0 -> u[ix + 1][iy][iz + 1];
00525
            u[6] = met0 -> u[ix][iy + 1][iz + 1];
           u[7] = met0 -> u[ix + 1][iy + 1][iz + 1];
00526
00527
00528
            v[0] = met0 -> v[ix][iy][iz];
            v[1] = met0 -> v[ix + 1][iy][iz];
00530
            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];
00531
00532
           v[5] = met0->v[ix + 1][iy][iz + 1];
v[6] = met0->v[ix][iy + 1][iz + 1];
00533
00534
00535
            v[7] = met0 -> v[ix + 1][iy + 1][iz + 1];
00536
00537
            w[0] = met0->w[ix][iy][iz];
00538
            w[1] = met0->w[ix + 1][iy][iz];
           w[2] = met0->w[ix][iy + 1][iz];
00539
           w[3] = met0->w[ix + 1][iy + 1][iz];
w[4] = met0->w[ix][iy][iz + 1];
00540
00542
            w[5] = met0 -> w[ix + 1][iy][iz + 1];
00543
            w[6] = met0 -> w[ix][iy + 1][iz + 1];
00544
            w[7] = met0 -> w[ix + 1][iy + 1][iz + 1];
00545
            /* Get indices... */
00546
           ix = locate(met1->lon, met1->nx, atm->lon[ip]);
00547
            iy = locate(met1->lat, met1->ny, atm->lat[ip]);
00548
00549
            iz = locate(met1->p, met1->np, atm->p[ip]);
00550
00551
            /* Collect local wind data... */
           u[8] = met1->u[ix][iy][iz];
00552
           u[9] = met1->u[ix + 1][iy][iz];
00553
            u[10] = met1->u[ix][iy + 1][iz];
           u[11] = met1->u[ix + 1][iy + 1][iz];
u[12] = met1->u[ix][iy][iz + 1];
00555
00556
           u[13] = met1->u[ix + 1][iy][iz + 1];

u[14] = met1->u[ix][iy + 1][iz + 1];
00557
00558
00559
           u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00561
            v[8] = met1->v[ix][iy][iz];
00562
            v[9] = met1 -> v[ix + 1][iy][iz];
00563
            v[10] = met1->v[ix][iy + 1][iz];
           v[11] = met1->v[ix + 1][iy + 1][iz];
v[12] = met1->v[ix][iy][iz + 1];
00564
00565
           v[13] = met1->v[ix + 1][iy][iz + 1];
v[14] = met1->v[ix][iy + 1][iz + 1];
00566
00567
00568
            v[15] = met1 - v[ix + 1][iy + 1][iz + 1];
00569
00570
            w[8] = met1->w[ix][iy][iz];
00571
           w[9] = met1->w[ix + 1][iy][iz];
           w(10) = met1->w(ix) [iy + 1] [iz];
w(11) = met1->w(ix + 1) [iy + 1] [iz];
w(12) = met1->w(ix) [iy] [iz + 1];
00572
00573
00574
00575
            w[13] = met1->w[ix + 1][iy][iz + 1];
00576
            w[14] = met1->w[ix][iy + 1][iz + 1];
00577
            w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00578
00579
            /* Get standard deviations of local wind data... */
00580
            usig = gsl_stats_sd(u, 1, 16);
00581
            vsig = gsl_stats_sd(v, 1, 16);
00582
            wsig = gsl_stats_sd(w, 1, 16);
00583
00584
            /\star Set temporal correlations for mesoscale fluctuations... \star/
00585
           r = 1 - 2 * fabs(dt) / ctl->dt_met;
00586
            rs = sqrt(1 - r * r);
00587
00588
            /\star Calculate mesoscale wind fluctuations... \star/
            atm->up[ip]
00589
00590
              r * atm->up[ip] + rs * gsl ran gaussian ziggurat(rng.
```

```
ctl->turb_meso * usig);
00592
            atm->vp[ip] =
00593
             r * atm->vp[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
                                                                              ctl->turb_meso * vsiq);
00594
00595
            atm->wp[ip] =
              r * atm->wp[ip] + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_meso * wsig);
00596
00598
            /* Calculate air parcel displacement... */
atm->lon[ip] += dx2deg(atm->up[ip] * dt / 1000., atm->lat[ip]);
atm->lat[ip] += dy2deg(atm->vp[ip] * dt / 1000.);
00599
00600
00601
            atm->p[ip] += atm->wp[ip] * dt;
00602
00603
00604 }
```

Here is the call graph for this function:



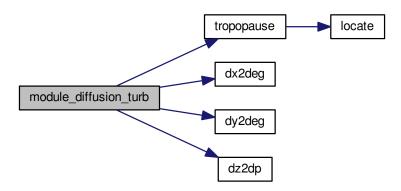
 $5.31.2.5 \quad \text{void module\_diffusion\_turb ( } \textbf{ctl\_t} * \textit{ctl, } \textbf{atm\_t} * \textit{atm, } \textbf{int } \textit{ip, } \textbf{double } \textit{dt, } \textbf{gsl\_rng} * \textit{rng } \textbf{)}$ 

Calculate turbulent diffusion.

Definition at line 608 of file trac.c.

```
00613
00614
00615
       double dx, dz, pt, p0, p1, w;
00616
00617
       /* Get tropopause pressure... */
00618
       pt = tropopause(atm->time[ip], atm->lat[ip]);
00619
00620
       /* Get weighting factor... */
       p1 = pt * 0.866877899;
p0 = pt / 0.866877899;
00621
00622
00623
       if (atm->p[ip] > p0)
00624
       else if (atm->p[ip] < p1)</pre>
00625
         w = 0;
00626
00627
       else
         w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00628
00629
00630
       /* Set diffusivitiy... */
       00631
00632
00633
00634
       /* Horizontal turbulent diffusion... */
00635
       if (dx > 0) {
00636
         atm->lon[ip]
00637
           += dx2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00638
                     / 1000., atm->lat[ip]);
         atm->lat[ip]
00639
00640
           += dy2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00641
                    / 1000.);
00642
```

Here is the call graph for this function:



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

Force air parcels to stay on isosurface.

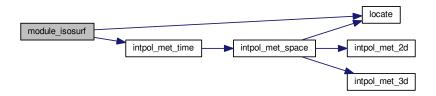
Definition at line 653 of file trac.c.

```
00658
                  {
00659
         static double *iso, *ps, t, *ts;
00661
        static int idx, ip2, n, nb = 100000;
00662
00663
00664
        FILE *in;
00665
00666
        char line[LEN];
00667
         /* Check control parameter... */ if (ctl->isosurf < 1 || ctl->isosurf > 4)
00668
00669
00670
           return:
00671
         /* Initialize... */
00673
        if (ip < 0) {</pre>
00674
00675
           /* Allocate... */
00676
           ALLOC(iso, double,
00677
                  NP);
           ALLOC(ps, double, nb);
00678
00679
00680
           ALLOC(ts, double,
00681
                 nb);
00682
           /* Save pressure... */
if (ctl->isosurf == 1)
00683
00684
00685
            for (ip2 = 0; ip2 < atm->np; ip2++)
00686
               iso[ip2] = atm->p[ip2];
00687
00688
           /* Save density... */
else if (ctl->isosurf == 2)
00689
00690
             for (ip2 = 0; ip2 < atm->np; ip2++) {
00691
                intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
```

```
atm->lon[ip2], atm->lat[ip2], NULL, &t, NULL, NULL,
00693
                                   NULL, NULL, NULL);
00694
                iso[ip2] = atm->p[ip2] / t;
00695
              }
00696
            /* Save potential temperature... */
00697
           else if (ctl->isosurf == 3)
00698
00699
              for (ip2 = 0; ip2 < atm->np; ip2++) {
00700
                intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
                atm->lon[ip2], atm->lat[ip2], NULL, &t, NULL, NULL, NULL, NULL, NULL); iso[ip2] = t * pow(P0 / atm->p[ip2], 0.286);
00701
00702
00703
00704
00705
00706
            /* Read balloon pressure data... */
00707
           else if (ctl->isosurf == 4) {
00708
00709
             /* Write info... */
printf("Read balloon pressure data: %s\n", ctl->balloon);
00710
00711
00712
              /* Open file... */
00713
              if (!(in = fopen(ctl->balloon, "r")))
               ERRMSG("Cannot open file!");
00714
00715
00716
              /* Read pressure time series... */
              while (fgets(line, LEM, in))
if (sscanf(line, "%lg %lg", &ts[n], &ps[n]) == 2)
00717
00718
                  if ((++n) > 100000)

ERRMSG("Too many data points!");
00719
00720
00721
00722
              /* Check number of points... */
00723
              if (n < 1)
00724
                ERRMSG("Could not read any data!");
00725
00726
              /* Close file... */
              fclose(in);
00727
00728
           }
00729
00730
           /* Leave initialization... */
00731
           return;
00732
00733
         /* Restore pressure... */
if (ctl->isosurf == 1)
00734
00735
00736
           atm \rightarrow p[ip] = iso[ip];
00737
         /* Restore density... */
else if (ctl->isosurf == 2) {
00738
00739
           intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00740
      lon[ip].
00741
                              atm->lat[ip], NULL, &t, NULL, NULL, NULL, NULL, NULL);
00742
           atm \rightarrow p[ip] = iso[ip] * t;
00743
00744
         /* Restore potential temperature... */
else if (ctl->isosurf == 3) {
00745
00746
           intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
       lon[ip],
           atm->lat[ip], NULL, &t, NULL, NULL, NULL, NULL, NULL); \\ atm->p[ip] = P0 * pow(iso[ip] / t, -1. / 0.286); \\
00748
00749
00750
00751
00752
         /* Interpolate pressure... */
00753
         else if (ctl->isosurf == 4)
00754
          if (atm->time[ip] <= ts[0])</pre>
00755
             atm->p[ip] = ps[0];
00756
           else if (atm->time[ip] >= ts[n-1])
00757
             atm->p[ip] = ps[n - 1];
00758
           else {
00759
             idx = locate(ts, n, atm->time[ip]);
              atm->p[ip] = LIN(ts[idx], ps[idx],
ts[idx + 1], ps[idx + 1], atm->time[ip]);
00760
00761
00762
00763
        }
00764 }
```

Here is the call graph for this function:



```
5.31.2.7 void module_meteo ( ctl t * ctl, met t * met0, met t * met1, atm t * atm, int ip )
```

Interpolate meteorological data for air parcel positions.

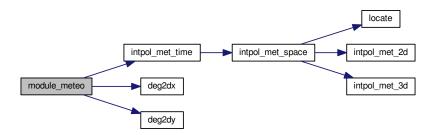
Definition at line 768 of file trac.c.

```
00773
00774
        double b, c, ps, p1, p_hno3, p_h2o, t, t1, term1, term2,
    u, u1, v, v1, w, x1, x2, h2o, o3, grad, vort;
00775
00777
00778
        /* Interpolate meteorological data... */
00779
        intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00780
                           atm->lat[ip], &ps, &t, &u, &v, &w, &h2o, &o3);
00781
00782
         /* Set surface pressure... */
00783
         if (ctl->qnt_ps >= 0)
00784
           atm->q[ctl->qnt_ps][ip] = ps;
00785
00786
        /* Set pressure... */
if (ctl->qnt_p >= 0)
00787
00788
           atm->q[ctl->qnt_p][ip] = atm->p[ip];
00789
00790
         /* Set temperature... */
00791
         if (ctl->qnt_t >= 0)
00792
           atm->q[ctl->qnt_t][ip] = t;
00793
00794
         /* Set zonal wind... */
00795
        if (ctl->qnt_u >= 0)
00796
           atm->q[ctl->qnt_u][ip] = u;
00797
00798
         /* Set meridional wind... */
00799
         if (ctl->qnt_v >= 0)
00800
          atm \rightarrow q[ctl \rightarrow qnt_v][ip] = v;
00801
00802
         /* Set vertical velocity... */
         if (ctl->qnt_w >= 0)
  atm->q[ctl->qnt_w][ip] = w;
00803
00804
00805
00806
         /* Set water vapor vmr... */
00807
         if (ctl->qnt_h2o >= 0)
00808
           atm->q[ctl->qnt\_h2o][ip] = h2o;
00809
         /* Set ozone vmr... */
00810
00811
         if (ctl->qnt_o3 >= 0)
00812
           atm->q[ctl->qnt_o3][ip] = o3;
00813
00814
         /* Calculate potential temperature... */
00815
         if (ctl->qnt_theta >= 0)
          atm->q[ctl->qnt\_theta][ip] = t * pow(P0 / atm->p[ip], 0.286);
00816
00817
00818
         /* Calculate potential vorticity... */
00819
         if (ctl->qnt_pv >= 0) {
00820
           /* Absolute vorticity... */
vort = 2 * 7.2921e-5 * sin(atm->lat[ip] * M_PI / 180.);
if (fabs(atm->lat[ip]) < 89.) {
00821
00822
00823
00824
             intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00825
                                (atm->lon[ip] >=
```

```
00826
                                     0 ? atm->lon[ip] - 1. : atm->lon[ip] + 1.),
00827
                                    atm->lat[ip], NULL, NULL, NULL, &v1, NULL, NULL, NULL);
00828
               vort += (v1 - v) / 1000.
00829
                 / ((atm->lon[ip] >= 0 ? -1 : 1) * deg2dx(1., atm->lat[ip]));
00830
00831
             intpol met time (met0, met1, atm->time(ip), atm->p(ip), atm->
       lon[ip],
00832
                                 (atm->lat[ip] >=
            0 ? atm->lat[ip] - 1. : atm->lat[ip] + 1.), NULL, NULL, 
&u1, NULL, NULL, NULL);
vort += (u1 - u) / 1000. / ((atm->lat[ip] >= 0 ? -1 : 1) * deg2dy(1.));
00833
00834
00835
00836
00837
             /* Potential temperature gradient... */
00838
            p1 = 0.85 * atm -> p[ip];
00839
             intpol_met_time(met0, met1, atm->time[ip], p1, atm->lon[ip],
            atm->lat[ip], NULL, &t1, NULL, NULL, NULL, NULL, NULL, NULL); grad = (t1 * pow(P0 / p1, 0.286) - t * pow(P0 / atm->p[ip], 0.286))
00840
00841
00842
               / (100. * (p1 - atm->p[ip]));
00843
00844
             /* Calculate PV... */
            atm \rightarrow q[ctl \rightarrow qnt_pv][ip] = -1e6 * G0 * vort * grad;
00845
00846
00847
          /* Calculate T_ice (Marti and Mauersberger, 1993)... */
if (ctl->qnt_tice >= 0 || ctl->qnt_tsts >= 0)
00848
00849
          atm \rightarrow q[ctl \rightarrow qnt\_tice][ip] = -2663.5
00851
               / (log10(ctl->psc_h2o * atm->p[ip] * 100.) - 12.537);
00852
00853
          /\star Calculate T_NAT (Hanson and Mauersberger, 1988)... \star/
          if (ctl->qnt_tnat >= 0 || ctl->qnt_tsts >= 0) {
   p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
   p_h2o = ctl->psc_h2o * atm->p[ip] / 1.333224;
   term1 = 38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o);
00854
00855
00856
00857
             term2 = 0.009179 - 0.00088 * log10(p_h2o);
00858
            b = term1 / term2;
c = -11397.0 / term2;
00859
00860
            x1 = (-b + sqrt(b * b - 4. * c)) / 2.;

x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
00861
00862
            if (x1 > 0)
00863
00864
              atm->q[ctl->qnt_tnat][ip] = x1;
00865
            if (x2 > 0)
00866
               atm->q[ctl->qnt_tnat][ip] = x2;
00867
00868
00869
          /* Calculate T_STS (mean of T_ice and T_NAT)... */
00870
          if (ctl->qnt_tsts >= 0) {
00871
          if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)</pre>
              ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
00872
            atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
00873
00874
                                                          + atm->q[ctl->qnt_tnat][ip]);
00875
          }
00876 }
```

Here is the call graph for this function:



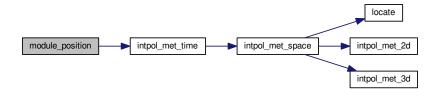
5.31.2.8 void module\_position (  $met_t * met0$ ,  $met_t * met1$ ,  $atm_t * atm$ , int ip )

Check position of air parcels.

Definition at line 880 of file trac.c.

```
00884
                    {
00885
00886
          double ps;
00887
00888
          /* Calculate modulo... */
         atm->lon[ip] = fmod(atm->lon[ip], 360);
atm->lat[ip] = fmod(atm->lat[ip], 360);
00889
00891
         /* Check latitude... */
while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
   if (atm->lat[ip] > 90) {
     atm->lat[ip] = 180 - atm->lat[ip];
}
00892
00893
00894
00895
               atm->lon[ip] += 180;
00896
00897
00898
            if (atm->lat[ip] < -90) {</pre>
               atm->lat[ip] = -180 - atm->lat[ip];
atm->lon[ip] += 180;
00899
00900
00901
00902
00903
         /* Check longitude... */
while (atm->lon[ip] < -180)</pre>
00904
00905
00906
           atm->lon[ip] += 360;
00907
          while (atm->lon[ip] >= 180)
00908
            atm->lon[ip] -= 360;
00909
00910
          /\star Get surface pressure... \star/
00911
         intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00912
                              atm->lon[ip], atm->lat[ip], &ps, NULL,
                              NULL, NULL, NULL, NULL);
00913
00914
00915
          /* Check pressure... */
00916
         if (atm->p[ip] > ps)
00917
            atm->p[ip] = ps;
         else if (atm->p[ip] < met0->p[met0->np - 1])
00918
            atm->p[ip] = met0->p[met0->np - 1];
00919
00920 }
```

Here is the call graph for this function:



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

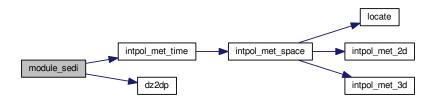
Calculate sedimentation of air parcels.

Definition at line 924 of file trac.c.

```
00930
00931
00932
        /\star Coefficients for Cunningham slip-flow correction (Kasten, 1968): \star/
00933
        const double A = 1.249, B = 0.42, C = 0.87;
00934
00935
        /\star Specific gas constant for dry air [J/(kg K)]: \star/
00936
        const double R = 287.058;
00937
00938
        /* Average mass of an air molecule [kg/molec]: */
00939
        const double m = 4.8096e-26;
00940
00941
        double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
00942
00943
        /* Check if parameters are available... */
```

```
if (ctl->qnt_r < 0 || ctl->qnt_rho < 0)</pre>
00945
          return;
00946
00947
         /* Convert units... */
         p = 100 * atm->p[ip];
00948
00949
         r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
        rho_p = atm->q[ctl->qnt_rho][ip];
00951
00952
         /* Get temperature... */
00953
        intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
                           atm->lat[ip], NULL, &T, NULL, NULL, NULL, NULL, NULL);
00954
00955
        /* Density of dry air... */
rho = p / (R * T);
00956
00957
00958
        /* Dynamic viscosity of air... */
eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
00959
00960
00961
         /* Thermal velocity of an air molecule... */ v = sqrt(8 * GSL_CONST_MKSA_BOLTZMANN * T / (M_PI * m));
00962
00963
00964
00965
         /\star Mean free path of an air molecule... \star/
00966
         lambda = 2 * eta / (rho * v);
00967
00968
         /* Knudsen number for air... */
00969
         K = lambda / r_p;
00970
00971
         /* Cunningham slip-flow correction... */
00972
         G = 1 + K * (A + B * exp(-C / K));
00973
00974
         /* Sedimentation (fall) velocity... */
00975
         v_p =
00976
           2. * gsl_pow_2(r_p) * (rho_p -
00977
                                      rho) * GSL_CONST_MKSA_GRAV_ACCEL / (9. * eta) * G;
00978
        /* Calculate pressure change... */
atm->p[ip] += dz2dp(v_p * dt / 1000., atm->p[ip]);
00979
00980
00981 }
```

Here is the call graph for this function:



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

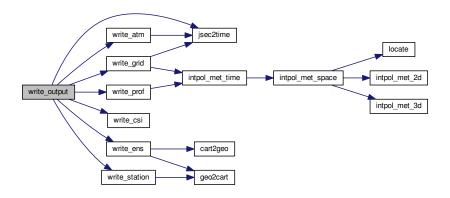
Write simulation output.

Definition at line 985 of file trac.c.

```
00991
                     {
00992
00993
         char filename[LEN]:
00994
00995
         double r;
00996
00997
         int year, mon, day, hour, min, sec;
00998
00999
         / \star \ \mathsf{Get} \ \mathsf{time...} \ \star /
01000
         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01001
         /* Write atmospheric data... */
```

```
if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
dirname, ctl->atm_basename, year, mon, day, hour, min);
01004
01005
          write_atm(filename, ctl, atm, t);
01006
01007
01008
01009
        /* Write CSI data... */
01010
        if (ctl->csi_basename[0] != '-') {
        sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01011
01012
          write_csi(filename, ctl, atm, t);
01013
01014
        /* Write ensemble data... */
if (ctl->ens_basename[0] != '-') {
01015
01016
01017
        sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01018
          write_ens(filename, ctl, atm, t);
01019
01020
01021
        /* Write gridded data... */
01022
        if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01023
         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01024
                  dirname, ctl->grid_basename, year, mon, day, hour, min);
          write_grid(filename, ctl, met0, met1, atm, t);
01025
01026
01027
01028
        /* Write profile data... */
        if (ctl->prof_basename[0] != '-') {
01029
        sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01030
01031
          write_prof(filename, ctl, met0, met1, atm, t);
01032
01033
01034
        /* Write station data...
01035
        if (ctl->stat_basename[0] != '-') {
01036
         sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01037
          write_station(filename, ctl, atm, t);
01038
01039 }
```

Here is the call graph for this function:



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

Definition at line 160 of file trac.c.

```
00162 {
00163
00164 ctl_t ctl;
00165 atm_t *atm;
00167
00168 met_t *met0, *met1;
00169
00170 gsl_rng *rng[NTHREADS];
```

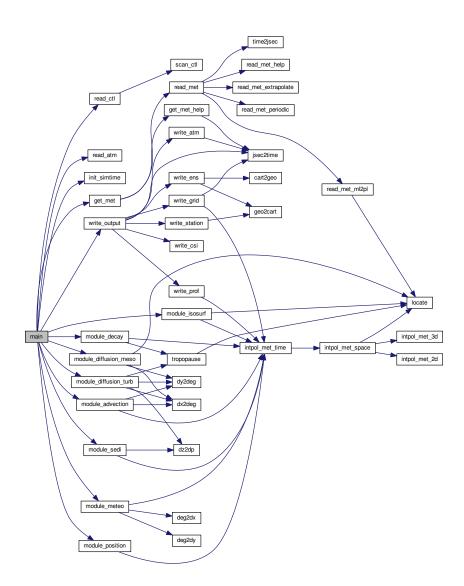
```
00172
        FILE *dirlist;
00173
        char dirname[LEN], filename[LEN];
00174
00175
00176
        double *dt, t, t0;
00177
00178
        int i, ip, ntask = 0, rank = 0, size = 1;
00179
00180 #ifdef MPI
00181
        /* Initialize MPI... */
        MPI_Init(&argc, &argv);
00182
        MPI_Comm_rank (MPI_COMM_WORLD, &rank);
00183
00184
        MPI_Comm_size(MPI_COMM_WORLD, &size);
00185 #endif
00186
00187
         /* Check arguments... */
        if (argc < 5)
00188
          ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00189
00190
00191
        /* Open directory list... */
00192
        if (!(dirlist = fopen(argv[1], "r")))
00193
          ERRMSG("Cannot open directory list!");
00194
        /* Loop over directories... */
while (fscanf(dirlist, "%s", dirname) != EOF) {
00195
00196
00197
00198
           /* MPI parallelization... */
00199
          if ((++ntask) % size != rank)
00200
            continue;
00201
00202
00203
             Initialize model run...
00204
00205
          /* Set timers... */
START_TIMER(TIMER_TOTAL);
00206
00207
00208
          START_TIMER(TIMER_INIT);
00210
           /* Allocate... */
00211
           ALLOC(atm, atm_t, 1);
00212
           ALLOC(met0, met_t, 1);
           ALLOC(met1, met_t, 1);
00213
          ALLOC(dt, double, NP);
00214
00215
00216
00217
           /\star Read control parameters... \star/
00218
           sprintf(filename, "%s/%s", dirname, argv[2]);
00219
           read_ctl(filename, argc, argv, &ctl);
00220
00221
           /* Initialize random number generators... */
00222
          gsl_rng_env_setup();
00223
           for (i = 0; i < NTHREADS; i++)</pre>
00224
            rng[i] = gsl_rng_alloc(gsl_rng_default);
00225
          /* Read atmospheric data... */
sprintf(filename, "%s/%s", dirname, argv[3]);
read_atm(filename, &ctl, atm);
00226
00227
00228
00229
00230
           /* Get simulation time interval... */
00231
           init_simtime(&ctl, atm);
00232
00233
           /* Get rounded start time... */
00234
           if (ctl.direction == 1)
00235
             t0 = floor(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00236
           else
00237
            t0 = ceil(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00238
00239
           /* Set timers... */
00240
           STOP_TIMER(TIMER_INIT);
00241
00242
00243
             Loop over timesteps...
00244
00245
          /* Loop over timesteps... */
for (t = t0; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;</pre>
00246
00247
00248
                t += ctl.direction * ctl.dt_mod) {
00249
            /* Adjust length of final time step... */ if (ctl.direction * (t - ctl.t_stop) > 0)
00250
00251
00252
               t = ctl.t stop;
00253
00254
             /* Set time steps for air parcels... */
00255
             for (ip = 0; ip < atm\rightarrownp; ip++)
              if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
00256
                    00257
00258
```

```
dt[ip] = t - atm->time[ip];
00260
00261
                 dt[ip] = GSL_NAN;
00262
00263
             /* Get meteorological data... */
00264
            START_TIMER(TIMER_INPUT);
00265
             get_met(&ctl, argv[4], t, met0, met1);
00266
             STOP_TIMER(TIMER_INPUT);
00267
            /* Initialize isosurface... */
START_TIMER(TIMER_ISOSURF);
00268
00269
00270
            if (t == t0)
  module_isosurf(&ctl, met0, met1, atm, -1);
00271
00272
             STOP_TIMER(TIMER_ISOSURF);
00273
00274
             /* Advection... */
            START_TIMER(TIMER_ADVECT);
00275
00276 #pragma omp parallel for default(shared) private(ip)
            for (ip = 0; ip < atm->np; ip++)
             if (gsl_finite(dt[ip]))
00278
00279
                module_advection(met0, met1, atm, ip, dt[ip]);
00280
            STOP_TIMER(TIMER_ADVECT);
00281
             /* Turbulent diffusion...
00282
00283
            START_TIMER(TIMER_DIFFTURB);
00284 #pragma omp parallel for default(shared) private(ip)
00285
            for (ip = 0; ip < atm->np; ip++)
00286
              if (gsl_finite(dt[ip]))
00287
                module_diffusion_turb(&ctl, atm, ip, dt[ip],
00288
                                         rng[omp_get_thread_num()]);
00289
            STOP_TIMER(TIMER_DIFFTURB);
00290
00291
             /* Mesoscale diffusion... */
00292
             START_TIMER (TIMER_DIFFMESO);
00293 \#pragma omp parallel for default(shared) private(ip)
00294
            for (ip = 0; ip < atm->np; ip++)
00295
              if (gsl_finite(dt[ip]))
                module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
00297
                                         rng[omp_get_thread_num()]);
00298
            STOP_TIMER(TIMER_DIFFMESO);
00299
00300
             /* Sedimentation...
            START_TIMER (TIMER SEDI):
00301
00302 #pragma omp parallel for default(shared) private(ip)
            for (ip = 0; ip < atm->np; ip++)
00303
00304
              if (gsl_finite(dt[ip]))
00305
                module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
00306
            STOP_TIMER(TIMER_SEDI);
00307
             /* Isosurface...
00308
            START_TIMER(TIMER_ISOSURF);
00309
00310 #pragma omp parallel for default(shared) private(ip)
00311
            for (ip = 0; ip < atm->np; ip++)
              module_isosurf(&ctl, met0, met1, atm, ip);
00312
00313
            STOP_TIMER(TIMER_ISOSURF);
00314
             /* Position... */
            START_TIMER(TIMER_POSITION);
00316
00317 #pragma omp parallel for default(shared) private(ip)
           for (ip = 0; ip < atm->np; ip++)
  module_position(met0, met1, atm, ip);
00318
00319
00320
            STOP TIMER (TIMER POSITION);
00321
00322
             /* Meteorological data... */
00323
            START_TIMER (TIMER_METEO);
00324 #pragma omp parallel for default(shared) private(ip)
00325
           for (ip = 0; ip < atm->np; ip++)
   module_meteo(&ctl, met0, met1, atm, ip);
00326
00327
            STOP TIMER (TIMER METEO):
00328
00329
             /* Decay... */
00330
            START_TIMER (TIMER_DECAY);
00331 \#pragma omp parallel for default(shared) private(ip)
            for (ip = 0; ip < atm->np; ip++)
00332
             if (gsl_finite(dt[ip]))
  module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00333
00334
00335
            STOP_TIMER(TIMER_DECAY);
00336
            /* Write output... */
START_TIMER(TIMER_OUTPUT);
00337
00338
00339
             write_output(dirname, &ctl, met0, met1, atm, t);
00340
            STOP_TIMER(TIMER_OUTPUT);
00341
00342
00343
             Finalize model run...
00344
00345
```

```
00346
            /* Report timers... */
STOP_TIMER(TIMER_TOTAL);
PRINT_TIMER(TIMER_TOTAL);
00347
00348
00349
            PRINT_TIMER(TIMER_INIT);
PRINT_TIMER(TIMER_INPUT);
PRINT_TIMER(TIMER_OUTPUT);
00350
00351
00352
00353
             PRINT_TIMER (TIMER_ADVECT);
00354
             PRINT_TIMER(TIMER_DECAY);
00355
             PRINT_TIMER(TIMER_DIFFMESO);
             PRINT_TIMER (TIMER_DIFFTURB);
PRINT_TIMER (TIMER_ISOSURF);
00356
00357
00358
             PRINT_TIMER (TIMER_METEO);
00359
             PRINT_TIMER (TIMER_POSITION);
00360
             PRINT_TIMER(TIMER_SEDI);
00361
            /* Report memory usage... */
printf("MEMORY_ATM = %g MByte\n", 2. * sizeof(atm_t) / 1024. / 1024.);
printf("MEMORY_METEO = %g MByte\n", 2. * sizeof(met_t) / 1024. / 1024.);
printf("MEMORY_DYNAMIC = %g MByte\n",
00362
00363
00364
00365
             00366
00367
00368
00369
00370
00371
00372
             /* 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());
00373
00374
00375
00376
00377
             /* Free random number generators... */
00378
             for (i = 0; i < NTHREADS; i++)</pre>
00379
               gsl_rng_free(rng[i]);
00380
00381
             /* Free... */
00382
             free(atm);
00383
             free (met0);
00384
             free (met1);
00385
             free(dt);
00386
00387
00388 #ifdef MPI
          /* Finalize MPI... */
00389
00390 MPI_Finalize();
00391 #endif
00392
00393
          return EXIT_SUCCESS;
00394 }
```

5.32 trac.c 177

Here is the call graph for this function:



## 5.32 trac.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
         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,
         but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
00012
         GNU General Public License for more details.
00013
         You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
00016
00017
         Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 #ifdef MPI
```

```
00028 #include "mpi.h"
00029 #endif
00030
00031 /* -----
        Defines...
00032
00033
00036 #define TIMER_TOTAL 0
00037
00039 #define TIMER_INIT 1
00040
00042 #define TIMER INPUT 2
00043
00045 #define TIMER_OUTPUT 3
00046
00048 #define TIMER_ADVECT 4
00049
00051 #define TIMER_DECAY 5
00052
00054 #define TIMER_DIFFMESO 6
00055
00057 #define TIMER_DIFFTURB 7
00058
00060 #define TIMER_ISOSURF 8
00061
00063 #define TIMER_METEO 9
00064
00066 #define TIMER_POSITION 10
00067
00069 #define TIMER SEDI 11
00070
00071 /* -
00072
       Functions...
00073
00074
00076 void init_simtime(
00077 ctl_t * ctl,
00078 atm_t * atm);
00079
00081 void module_advection(
00082
        met_t * met0,
        met_t * met1,
00083
        atm_t * atm, int ip,
00084
00085
00086
        double dt);
00087
00089 void module_decay(
00090
        ctl_t * ctl,
00091
        met_t * met0,
        met_t * met1,
atm_t * atm,
00092
00093
00094
        int ip,
00095
        double dt);
00096
00098 void module_diffusion_meso(
        ctl_t * ctl,
met_t * met0,
00099
00100
00101
        met_t * met1,
        atm_t * atm,
00102
00103
        int ip,
        double dt,
00104
00105
        gsl_rng * rng);
00106
00108 void module_diffusion_turb(
        ctl_t * ctl,
atm_t * atm,
00109
00110
00111
        int ip,
00112
        double dt.
00113
        gsl_rng * rng);
00114
00116 void module_isosurf(
00117
        ctl_t * ctl,
        met_t * met0,
00118
        met_t * met1,
atm_t * atm,
00119
00120
00121
        int ip);
00122
00124 void module_meteo(
        ctl_t * ctl,
met_t * met0,
00125
00126
        met_t * met1,
atm_t * atm,
00127
00128
00129
        int ip);
00130
00132 void module_position(
00133 met_t * met0,
00134 met_t * met1,
```

```
00135
        atm_t * atm,
00136
        int ip);
00137
00139 void module_sedi(
00140
        ctl_t * ctl,
met_t * met0,
00141
        met_t * met1,
00142
00143
        atm_t * atm,
00144
        int ip,
00145
        double dt);
00146
00148 void write_output(
00149
        const char *dirname,
        ctl_t * ctl,
met_t * met0,
00150
00151
        met_t * met1,
atm_t * atm,
00152
00153
00154
        double t);
00155
00156 /*
00157
00158
00159
00160 int main(
00161
        int argc,
00162
        char *argv[]) {
00163
00164
        ctl_t ctl;
00165
00166
        atm_t *atm;
00167
00168
        met_t *met0, *met1;
00169
00170
        gsl_rng *rng[NTHREADS];
00171
        FILE *dirlist:
00172
00173
00174
        char dirname[LEN], filename[LEN];
00175
00176
        double *dt, t, t0;
00177
00178
        int i, ip, ntask = 0, rank = 0, size = 1;
00179
00180 #ifdef MPI
        /* Initialize MPI... */
00181
00182
        MPI_Init(&argc, &argv);
00183
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00184
        MPI_Comm_size(MPI_COMM_WORLD, &size);
00185 #endif
00186
00187
         /* Check arguments... */
00188
        if (argc < 5)
00189
          ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00190
        /* Open directory list... */
if (!(dirlist = fopen(argv[1], "r")))
00191
00192
00193
          ERRMSG("Cannot open directory list!");
00194
        /* Loop over directories... */
while (fscanf(dirlist, "%s", dirname) != EOF) {
00195
00196
00197
           /\star MPI parallelization... \star/
00198
00199
          if ((++ntask) % size != rank)
00200
            continue;
00201
00202
00203
             Initialize model run...
00204
00205
00206
           /* Set timers... */
00207
           START_TIMER(TIMER_TOTAL);
00208
           START_TIMER(TIMER_INIT);
00209
00210
           /* Allocate... */
          ALLOC(atm, atm_t, 1);
ALLOC(met0, met_t, 1);
00211
00212
00213
           ALLOC(met1, met_t, 1);
00214
           ALLOC(dt, double,
00215
                 NP);
00216
00217
           /\star Read control parameters... \star/
           sprintf(filename, "%s/%s", dirname, argv[2]);
00218
00219
           read_ctl(filename, argc, argv, &ctl);
00220
00221
           /\star Initialize random number generators... \star/
           gsl_rng_env_setup();
for (i = 0; i < NTHREADS; i++)</pre>
00222
00223
```

```
00224
            rng[i] = gsl_rng_alloc(gsl_rng_default);
00225
          /* Read atmospheric data... */
sprintf(filename, "%s/%s", dirname, argv[3]);
read_atm(filename, &ctl, atm);
00226
00227
00228
00229
00230
           /* Get simulation time interval... */
00231
          init_simtime(&ctl, atm);
00232
00233
           /* Get rounded start time... */
00234
          if (ctl.direction == 1)
00235
            t0 = floor(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00236
          else
            t0 = ceil(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00237
00238
00239
           /* Set timers... */
00240
          STOP_TIMER(TIMER_INIT);
00241
00242
00243
             Loop over timesteps...
00244
00245
          /* Loop over timesteps... */
for (t = t0; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;</pre>
00246
00247
00248
                t += ctl.direction * ctl.dt_mod) {
00250
             /\star Adjust length of final time step... \star/
00251
             if (ctl.direction * (t - ctl.t_stop) > 0)
00252
               t = ctl.t_stop;
00253
00254
             /\star Set time steps for air parcels... \star/
00255
            for (ip = 0; ip < atm->np; ip++)
00256
              if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
                    00257
00258
00259
                 dt[ip] = t - atm->time[ip];
00260
              else
00261
                dt[ip] = GSL_NAN;
00262
00263
             /* Get meteorological data... */
00264
            START_TIMER(TIMER_INPUT);
            get_met(&ctl, argv[4], t, met0, met1);
STOP_TIMER(TIMER_INPUT);
00265
00266
00267
             /* Initialize isosurface... */
00268
00269
             START_TIMER (TIMER_ISOSURF);
            if (t == t0)
  module_isosurf(&ctl, met0, met1, atm, -1);
00270
00271
            STOP_TIMER(TIMER_ISOSURF);
00272
00273
             /* Advection... */
00275
            START_TIMER(TIMER_ADVECT);
00276 \#pragma omp parallel for default(shared) private(ip)
00277
           for (ip = 0; ip < atm->np; ip++)
             if (gsl_finite(dt[ip]))
  module_advection(met0, met1, atm, ip, dt[ip]);
00278
00279
             STOP_TIMER(TIMER_ADVECT);
00280
00281
00282
             /* Turbulent diffusion..
00283
            START_TIMER(TIMER_DIFFTURB);
00284 #pragma omp parallel for default(shared) private(ip)
            for (ip = 0; ip < atm->np; ip++)
00285
00286
              if (gsl_finite(dt[ip]))
                module_diffusion_turb(&ctl, atm, ip, dt[ip],
00287
00288
                                        rng[omp_get_thread_num()]);
00289
            STOP_TIMER(TIMER_DIFFTURB);
00290
00291
             /* Mesoscale diffusion... */
00292
            START_TIMER(TIMER_DIFFMESO);
00293 #pragma omp parallel for default(shared) private(ip)
00294
            for (ip = 0; ip < atm->np; ip++)
00295
              if (gsl_finite(dt[ip]))
00296
                module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
00297
                                        rng[omp_get_thread_num()]);
00298
            STOP_TIMER(TIMER_DIFFMESO);
00299
00300
             /* Sedimentation...
00301
            START_TIMER(TIMER_SEDI);
00302 \#pragma omp parallel for default(shared) private(ip)
            for (ip = 0; ip < atm->np; ip++)
00303
             if (gsl_finite(dt[ip]))
  module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
00304
00305
00306
            STOP TIMER (TIMER SEDI);
00307
             /* Isosurface... */
00308
            START_TIMER(TIMER_ISOSURF);
00309
00310 #pragma omp parallel for default(shared) private(ip)
```

```
for (ip = 0; ip < atm->np; ip++)
00312
              module_isosurf(&ctl, met0, met1, atm, ip);
00313
            STOP_TIMER(TIMER_ISOSURF);
00314
00315
            /* Position...
            START_TIMER(TIMER_POSITION);
00316
00317 #pragma omp parallel for default(shared) private(ip)
00318
           for (ip = 0; ip < atm->np; ip++)
              module_position(met0, met1, atm, ip);
00319
00320
            STOP_TIMER(TIMER_POSITION);
00321
            /* Meteorological data... */
00322
00323
            START_TIMER(TIMER_METEO);
00324 #pragma omp parallel for default(shared) private(ip)
00325
           for (ip = 0; ip < atm->np; ip++)
00326
              module_meteo(&ctl, met0, met1, atm, ip);
00327
            STOP_TIMER(TIMER_METEO);
00328
            /* Decay... */
00329
            START_TIMER(TIMER_DECAY);
00330
00331 #pragma omp parallel for default(shared) private(ip)
00332
            for (ip = 0; ip < atm->np; ip++)
             if (gsl_finite(dt[ip]))
00333
                module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00334
00335
            STOP_TIMER(TIMER_DECAY);
00336
00337
            /* Write output... */
00338
            START_TIMER(TIMER_OUTPUT);
            write_output(dirname, &ctl, met0, met1, atm, t);
STOP_TIMER(TIMER_OUTPUT);
00339
00340
00341
00342
00343
00344
            Finalize model run...
00345
00346
00347
          /* Report timers...
          STOP_TIMER(TIMER_TOTAL);
00349
          PRINT_TIMER (TIMER_TOTAL);
00350
          PRINT_TIMER(TIMER_INIT);
00351
          PRINT_TIMER(TIMER_INPUT);
          PRINT_TIMER(TIMER_OUTPUT);
00352
00353
          PRINT TIMER (TIMER ADVECT);
00354
          PRINT_TIMER (TIMER_DECAY);
00355
          PRINT_TIMER (TIMER_DIFFMESO);
00356
          PRINT_TIMER(TIMER_DIFFTURB);
00357
          PRINT_TIMER(TIMER_ISOSURF);
00358
          PRINT_TIMER (TIMER_METEO);
          PRINT_TIMER (TIMER_POSITION);
00359
00360
          PRINT TIMER (TIMER SEDI);
00361
00362
          /* Report memory usage... */
          00363
00364
00365
00366
00367
                 (((EX + EY) + (2 + NQ) * GX * GY * GZ) * sizeof(double)
+ (EX * EY + EX * EY * EP) * sizeof(float)
+ (2 * GX * GY * GZ) * sizeof(int)) / 1024. / 1024.);
00368
00369
00370
00371
          /* Report problem size... */
printf("SIZE_NP = %d\n", atm->np);
00372
00373
00374
          printf("SIZE_TASKS = %d\n", size);
          printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00375
00376
00377
          /* Free random number generators... */
for (i = 0; i < NTHREADS; i++)</pre>
00378
            gsl_rng_free(rng[i]);
00379
00380
00381
          /* Free... */
00382
          free (atm);
00383
          free (met0);
00384
          free (met1);
00385
          free(dt);
00386
00387
00388 #ifdef MPI
00389
        /* Finalize MPI... */
       MPI_Finalize();
00390
00391 #endif
00392
00393
        return EXIT_SUCCESS;
00394 }
00395
00397
```

```
00398 void init_simtime(
00399
      ctl_t * ctl,
       atm_t * atm) {
00400
00401
00402
       /* Set inital and final time... */
       if (ctl->direction == 1)
00403
        if (ctl->t_start < -1e99)</pre>
00405
           ctl->t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00406
         if (ctl->t\_stop < -1e99)
          ctl->t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00407
       } else if (ctl->direction == -1) {
00408
        if (ctl->t_stop < -1e99)
00409
          ctl->t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00410
00411
         if (ctl->t_start < -1e99)
00412
           ctl->t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00413
00414
00415
       /* Check time... */
       if (ctl->direction * (ctl->t_stop - ctl->t_start) <= 0)</pre>
00416
00417
         ERRMSG("Nothing to do!");
00418 }
00419
00421
00422 void module_advection(
00423
       met_t * met0,
00424
       met_t * met1,
00425
       atm_t * atm,
00426
       int ip,
       double dt) {
00427
00428
00429
       double v[3], xm[3];
00430
00431
       /* Interpolate meteorological data... */
       00432
00433
                      &v[0], &v[1], &v[2], NULL, NULL);
00434
00436
       /* Get position of the mid point... */
       00437
00438
       xm[2] = atm - p[ip] + 0.5 * dt * v[2];
00439
00440
00441
       /* Interpolate meteorological data for mid point... */
00442
       intpol_met_time(met0, met1, atm->time[ip] + 0.5 * dt,
00443
                      xm[2], xm[0], xm[1], NULL, NULL,
00444
                      &v[0], &v[1], &v[2], NULL, NULL);
00445
00446
       /* 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.);
00447
00448
00449
00450
       atm->p[ip] += dt * v[2];
00451 }
00452
00454
00455 void module_decay(
00456
       ctl_t * ctl,
       met_t * met0,
00457
       met_t * met1,
atm_t * atm,
00458
00459
00460
       int ip,
00461
       double dt) {
00462
00463
       double ps, pt, tdec;
00464
       /* Check lifetime values... */
00465
       if ((ctl->tdec_trop <= 0 && ctl->tdec_strat <= 0) || ctl->
00466
     qnt_m < 0
00467
00468
00469
       /* Set constant lifetime... */
       if (ctl->tdec_trop == ctl->tdec_strat)
  tdec = ctl->tdec_trop;
00470
00471
00472
00473
       /* Set altitude-dependent lifetime... */
00474
00475
00476
         /* Get surface pressure... */
00477
         intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
                        atm->lon[ip], atm->lat[ip], &ps, NULL,
00478
00479
                        NULL, NULL, NULL, NULL, NULL);
00480
00481
         /* Get tropopause pressure... */
00482
         pt = tropopause(atm->time[ip], atm->lat[ip]);
00483
```

```
/* Set lifetime... */
00485
           if (atm->p[ip] \le pt)
00486
             tdec = ctl->tdec_strat;
00487
           else
00488
            tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
      p[ip]);
00489
00490
00491
         /* Calculate exponential decay... */
00492
         atm->q[ctl->qnt_m][ip] *= exp(-dt / tdec);
00493 }
00494
00496
00497 void module_diffusion_meso(
        ctl_t * ctl,
met_t * met0,
00498
00499
00500
         met_t * met1,
         atm_t * atm,
00501
00502
         int ip,
00503
         double dt,
00504
         gsl_rng * rng) {
00505
00506
        double r, rs, u[16], v[16], w[16], usig, vsig, wsig;
00507
00508
        int ix, iy, iz;
00509
00510
         /* Calculate mesoscale velocity fluctuations... */
00511
         if (ctl->turb_meso > 0) {
00512
00513
           /* Get indices... */
00514
           ix = locate(met0->lon, met0->nx, atm->lon[ip]);
00515
           iy = locate(met0->lat, met0->ny, atm->lat[ip]);
00516
           iz = locate(met0->p, met0->np, atm->p[ip]);
00517
00518
           /* Collect local wind data... */
           u[0] = met0->u[ix][iy][iz];
u[1] = met0->u[ix + 1][iy][iz];
00519
00521
           u[2] = met0 -> u[ix][iy + 1][iz];
           u[3] = met0->u[ix + 1][iy + 1][iz];
u[4] = met0->u[ix][iy][iz + 1];
00522
00523
           u[5] = met0 -> u[ix + 1][iy][iz + 1];

u[6] = met0 -> u[ix][iy + 1][iz + 1];
00524
00525
00526
           u[7] = met0 -> u[ix + 1][iy + 1][iz + 1];
00527
           v[0] = met0 -> v[ix][iy][iz];
00528
00529
           v[1] = met0 -> v[ix + 1][iy][iz];
00530
           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];
00531
00532
           v[5] = met0 -> v[ix + 1][iy][iz + 1];
00534
           v[6] = met0 -> v[ix][iy + 1][iz + 1];
00535
           v[7] = met0 -> v[ix + 1][iy + 1][iz + 1];
00536
           w[0] = met0->w[ix][iy][iz];
00537
           w[1] = met0->w[ix + 1][iy][iz];
w[2] = met0->w[ix][iy + 1][iz];
00538
           w[3] = met0->w[ix + 1][iy + 1][iz];
w[4] = met0->w[ix][iy][iz + 1];
00540
00541
00542
           w[5] = met0->w[ix + 1][iy][iz + 1];
00543
           w[6] = met0->w[ix][iy + 1][iz + 1];
           w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00544
00545
00546
00547
           ix = locate(met1->lon, met1->nx, atm->lon[ip]);
00548
           iy = locate(met1->lat, met1->ny, atm->lat[ip]);
           iz = locate(met1->p, met1->np, atm->p[ip]);
00549
00550
00551
           /* Collect local wind data... */
           u[8] = met1->u[ix][iy][iz];
00553
           u[9] = met1 -> u[ix + 1][iy][iz];
00554
           u[10] = met1->u[ix][iy + 1][iz];
           u[11] = met1->u[ix + 1][iy + 1][iz];
u[12] = met1->u[ix][iy][iz + 1];
00555
00556
           u[13] = met1->u[ix + 1][iy][iz + 1];
u[14] = met1->u[ix][iy + 1][iz + 1];
00557
00558
00559
           u[15] = met1 -> u[ix + 1][iy + 1][iz + 1];
00560
           v[8] = met1->v[ix][iy][iz];
v[9] = met1->v[ix + 1][iy][iz];
00561
00562
           v[10] = met1->v[ix][iy + 1][iz];
00563
           v[11] = met1->v[ix + 1][iy + 1][iz];
v[12] = met1->v[ix][iy][iz + 1];
00564
00565
00566
           v[13] = met1 -> v[ix + 1][iy][iz + 1];
00567
           v[14] = met1->v[ix][iy + 1][iz + 1];
           v[15] = met1 -> v[ix + 1][iy + 1][iz + 1];
00568
00569
```

```
w[8] = met1->w[ix][iy][iz];
00571
          w[9] = met1->w[ix + 1][iy][iz];
00572
          w[10] = met1->w[ix][iy + 1][iz];
          w[11] = met1->w[ix + 1][iy + 1][iz];
w[12] = met1->w[ix][iy][iz + 1];
00573
00574
00575
          w[13] = met1 -> w[ix + 1][iy][iz + 1];
00576
          w[14] = met1->w[ix][iy + 1][iz + 1];
00577
          w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00578
00579
          /\star Get standard deviations of local wind data... \star/
00580
          usig = gsl_stats_sd(u, 1, 16);
00581
          vsig = gsl_stats_sd(v, 1, 16);
00582
          wsig = gsl_stats_sd(w, 1, 16);
00583
00584
          /\star Set temporal correlations for mesoscale fluctuations... \star/
          r = 1 - 2 * fabs(dt) / ctl->dt_met;
rs = sqrt(1 - r * r);
00585
00586
00587
00588
          /\star Calculate mesoscale wind fluctuations... \star/
00589
          atm->up[ip] =
00590
           r * atm->up[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
00591
                                                                ctl->turb_meso * usig);
00592
          atm->vp[ip] =
00593
           r * atm->vp[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
00594
                                                               ctl->turb_meso * vsig);
00595
          atm->wp[ip] =
00596
           r * atm->wp[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
00597
                                                               ctl->turb_meso * wsig);
00598
          /* Calculate air parcel displacement... */
atm->lon[ip] += dx2deg(atm->up[ip] * dt / 1000., atm->lat[ip]);
atm->lat[ip] += dy2deg(atm->vp[ip] * dt / 1000.);
00599
00600
00601
00602
          atm->p[ip] += atm->wp[ip] * dt;
00603
00604 }
00605
00607
00608 void module_diffusion_turb(
       ctl_t * ctl,
atm_t * atm,
00609
00610
00611
        int ip,
00612
        double dt.
00613
        gsl_rng * rng) {
00614
00615
        double dx, dz, pt, p0, p1, w;
00616
00617
        /* Get tropopause pressure... */
        pt = tropopause(atm->time[ip], atm->lat[ip]);
00618
00619
00620
        /* Get weighting factor... */
        p1 = pt * 0.866877899;
p0 = pt / 0.866877899;
00621
00622
00623
        if (atm->p[ip] > p0)
00624
          w = 1;
00625
        else if (atm->p[ip] < p1)</pre>
00626
00627
        else
00628
         w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00629
00630
        /* Set diffusivitiy... */
        dx = w * ctl - turb_dx_trop + (1 - w) * ctl - turb_dx_strat;
00631
00632
        dz = w * ctl - turb_dz_trop + (1 - w) * ctl - turb_dz_strat;
00633
00634
        /* Horizontal turbulent diffusion... */
00635
        if (dx > 0) {
00636
         atm->lon[ip]
00637
            += dx2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00638
                       / 1000., atm->lat[ip]);
00639
          atm->lat[ip]
00640
           += dy2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00641
                       / 1000.);
00642
00643
00644
        /* Vertical turbulent diffusion... */
00645
        if (dz > 0)
00646
         atm->p[ip]
00647
            += dz2dp(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dz * fabs(dt)))
00648
                      / 1000., atm->p[ip]);
00649 }
00650
00652
00653 void module_isosurf(
00654 ctl_t * ctl,
00655 met_t * met0,
00656
       met t * met1.
```

```
atm_t * atm,
00658
        int ip) {
00659
00660
        static double *iso, *ps, t, *ts;
00661
00662
        static int idx, ip2, n, nb = 100000;
00663
00664
        FILE *in;
00665
00666
        char line[LEN];
00667
        /* Check control parameter... */
if (ctl->isosurf < 1 || ctl->isosurf > 4)
00668
00669
00670
          return;
00671
00672
        /* Initialize... */
        if (ip < 0) {</pre>
00673
00674
00675
           /* Allocate... */
00676
          ALLOC(iso, double,
00677
                 NP);
00678
          ALLOC(ps, double,
00679
                nb);
00680
          ALLOC(ts, double,
00681
                nb);
00682
00683
           /* Save pressure... */
00684
           if (ctl->isosurf == 1)
00685
            for (ip2 = 0; ip2 < atm->np; ip2++)
00686
               iso[ip2] = atm->p[ip2];
00687
00688
           /* Save density... */
00689
          else if (ctl->isosurf == 2)
00690
             for (ip2 = 0; ip2 < atm->np; ip2++) {
00691
               intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
                                atm->lon[ip2], atm->lat[ip2], NULL, &t, NULL, NULL,
00692
                                NULL, NULL, NULL);
00693
               iso[ip2] = atm->p[ip2] / t;
00694
00695
00696
00697
           /* Save potential temperature... */
00698
           else if (ctl->isosurf == 3)
            for (ip2 = 0; ip2 < atm->np; ip2++) {
   intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00699
00700
00701
                                atm->lon[ip2], atm->lat[ip2], NULL, &t, NULL, NULL,
00702
                                NULL, NULL, NULL);
00703
               iso[ip2] = t * pow(P0 / atm->p[ip2], 0.286);
00704
00705
00706
           /* Read balloon pressure data... */
00707
          else if (ctl->isosurf == 4) {
00708
00709
             /\star Write info... \star/
00710
            printf("Read balloon pressure data: %s\n", ctl->balloon);
00711
00712
             /* Open file... */
00713
             if (!(in = fopen(ctl->balloon, "r")))
00714
               ERRMSG("Cannot open file!");
00715
00716
             /\star Read pressure time series... \star/
            while (fgets(line, LEN, in))
  if (sscanf(line, "%lg %lg", &ts[n], &ps[n]) == 2)
00717
00718
00719
                 if ((++n) > 100000)
00720
                   ERRMSG("Too many data points!");
00721
00722
             /\star Check number of points... \star/
00723
             if (n < 1)
               ERRMSG("Could not read any data!");
00724
00725
00726
             /* Close file... */
00727
            fclose(in);
00728
00729
00730
          /* Leave initialization... */
00731
          return;
00732
00733
00734
         /* Restore pressure... */
00735
        if (ctl->isosurf == 1)
00736
          atm->p[ip] = iso[ip];
00737
        /* Restore density... */
else if (ctl->isosurf == 2) {
00738
00739
00740
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00741
                            atm->lat[ip], NULL, &t, NULL, NULL, NULL, NULL, NULL);
00742
          atm->p[ip] = iso[ip] * t;
```

```
00743
        }
00744
00745
        /* Restore potential temperature... */
00746
        else if (ctl->isosurf == 3) {
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00747
      lon[ip],
          atm->p[ip] = P0 * pow(iso[ip] / t, -1. / 0.286);
00748
00749
00750
00751
00752
        /* Interpolate pressure... */
        else if (ctl->isosurf == 4) {
  if (atm->time[ip] <= ts[0])</pre>
00753
00754
00755
            atm->p[ip] = ps[0];
00756
          else if (atm->time[ip] >= ts[n - 1])
00757
            atm->p[ip] = ps[n - 1];
00758
          else {
00759
            idx = locate(ts, n, atm->time[ip]);
            atm->p[ip] = LIN(ts[idx], ps[idx],
ts[idx + 1], ps[idx + 1], atm->time[ip]);
00760
00761
00762
00763
        }
00764 }
00765
00767
00768 void module_meteo(
00769
        ctl_t * ctl,
        met_t * met0,
00770
00771
        met_t * met1,
atm_t * atm,
00772
00773
        int ip) {
00774
00775
        double b, c, ps, p1, p_hno3, p_h2o, t, t1, term1, term2,
00776
          u, u1, v, v1, w, x1, x2, h2o, o3, grad, vort;
00777
00778
        /* Interpolate meteorological data... */
intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00779
     lon[ip],
00780
                         atm->lat[ip], &ps, &t, &u, &v, &w, &h2o, &o3);
00781
00782
        /* Set surface pressure... */
        if (ctl->qnt_ps >= 0)
  atm->q[ctl->qnt_ps][ip] = ps;
00783
00784
00785
00786
        /* Set pressure...
00787
        if (ctl->qnt_p >= 0)
00788
         atm->q[ctl->qnt_p][ip] = atm->p[ip];
00789
00790
        /* Set temperature... */
00791
        if (ctl->qnt_t >= 0)
00792
          atm->q[ctl->qnt_t][ip] = t;
00793
        /* Set zonal wind... */
00794
00795
        if (ctl->qnt_u >= 0)
         atm->q[ctl->qnt_u][ip] = u;
00796
00797
00798
        /* Set meridional wind... */
00799
        if (ctl->qnt_v >= 0)
00800
          atm->q[ctl->qnt_v][ip] = v;
00801
        /* Set vertical velocity... */
if (ctl->qnt_w >= 0)
00802
00803
00804
          atm->q[ctl->qnt_w][ip] = w;
00805
00806
        /* Set water vapor vmr... */
00807
        if (ct1->qnt h2o >= 0)
          atm \rightarrow q[ctl \rightarrow qnt_h2o][ip] = h2o;
00808
00809
00810
        /* Set ozone vmr...
00811
        if (ctl->qnt_o3 >= 0)
00812
          atm->q[ctl->qnt_o3][ip] = o3;
00813
00814
        /\star Calculate potential temperature... \star/
00815
        if (ctl->gnt theta >= 0)
00816
          atm \rightarrow q[ctl \rightarrow qnt\_theta][ip] = t * pow(P0 / atm \rightarrow p[ip], 0.286);
00817
00818
        /* Calculate potential vorticity... */
00819
        if (ctl->qnt_pv >= 0) {
00820
00821
          /* Absolute vorticity... */
00822
          vort = 2 * 7.2921e-5 * sin(atm->lat[ip] * M_PI / 180.);
00823
          if (fabs(atm->lat[ip]) < 89.) {</pre>
00824
             intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00825
                             (atm->lon[ip] >=
                               0 ? atm->lon[ip] - 1. : atm->lon[ip] + 1.),
00826
                             atm->lat[ip], NULL, NULL, NULL, &v1, NULL, NULL, NULL);
00827
```

```
vort += (v1 - v) / 1000.
               / ((atm->lon[ip] >= 0 ? -1 : 1) * deg2dx(1., atm->lat[ip]));
00829
00830
00831
           intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00832
                              (atm->lat[ip] >=
                             0 ? atm->lat[ip] - 1. : atm->lat[ip] + 1.), NULL, NULL, &ul, NULL, NULL, NULL, NULL);
00833
00834
00835
           vort += (u1 - u) / 1000. / ((atm->lat[ip] >= 0 ? -1 : 1) * deg2dy(1.));
00836
00837
           /* Potential temperature gradient... */
           p1 = 0.85 * atm->p[ip];
00838
           00839
00840
00841
           grad = (t1 * pow(P0 / p1, 0.286) - t * pow(P0 / atm->p[ip], 0.286))
00842
             / (100. * (p1 - atm->p[ip]));
00843
00844
           /* Calculate PV... */
           atm \rightarrow q[ctl \rightarrow qnt_pv][ip] = -1e6 * G0 * vort * grad;
00845
00846
00847
00848
         /* Calculate T_ice (Marti and Mauersberger, 1993)... */
         if (ctl->qnt_tice >= 0 || ctl->qnt_tsts >= 0)
  atm->q[ctl->qnt_tice][ip] = -2663.5
00849
00850
00851
             / (log10(ctl->psc_h2o * atm->p[ip] * 100.) - 12.537);
00852
00853
         /* Calculate T_NAT (Hanson and Mauersberger, 1988)... \star/
00854
         if (ctl->qnt_tnat >= 0 || ctl->qnt_tsts >= 0)
00855
           p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
           p_h2o = ctl->psc_h2o * atm->p[ip] / 1.333224;
term1 = 38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o);
term2 = 0.009179 - 0.00088 * log10(p_h2o);
00856
00857
00858
00859
           b = term1 / term2;
00860
           c = -11397.0 / term2;
           x1 = (-b + sqrt(b * b - 4. * c)) / 2.;

x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
00861
00862
00863
           if (x1 > 0)
00864
             atm->q[ctl->qnt_tnat][ip] = x1;
00865
           if (x2 > 0)
00866
             atm->q[ctl->qnt_tnat][ip] = x2;
00867
00868
00869
         /\star Calculate T_STS (mean of T_ice and T_NAT)... \star/
00870
         if (ctl->qnt_tsts >= 0) {
00871
          if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)</pre>
00872
             ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
00873
           atm \rightarrow q[ctl \rightarrow qnt\_tsts][ip] = 0.5 * (atm \rightarrow q[ctl \rightarrow qnt\_tice][ip]
00874
                                                   + atm->q[ctl->qnt_tnat][ip]);
00875
00876 }
00877
00879
00880 void module position(
00881
        met_t * met0,
met_t * met1,
00882
        atm_t * atm,
00884
        int ip) {
00885
00886
         double ps;
00887
        /* Calculate modulo... */
atm->lon[ip] = fmod(atm->lon[ip], 360);
atm->lat[ip] = fmod(atm->lat[ip], 360);
00888
00889
00890
00891
        /* Check latitude... */
while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
   if (atm->lat[ip] > 90) {
     atm->lat[ip] = 180 - atm->lat[ip];
00892
00893
00894
00895
             atm->lon[ip] += 180;
00896
00897
00898
           if (atm->lat[ip] < -90) {
             atm->lat[ip] = -180 - atm->lat[ip];
atm->lon[ip] += 180;
00899
00900
00901
00902
00903
00904
         /* Check longitude... */
         while (atm->lon[ip] < -180)
00905
          atm->lon[ip] += 360;
00906
         while (atm->lon[ip] >= 180)
00907
00908
           atm->lon[ip] -= 360;
00909
00910
         /* Get surface pressure... */
00911
         intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00912
                           atm->lon[ip], atm->lat[ip], &ps, NULL,
                           NULL, NULL, NULL, NULL, NULL);
00913
```

```
00914
00915
        /* Check pressure... */
00916
        if (atm->p[ip] > ps)
00917
         atm->p[ip] = ps;
        else if (atm->p[ip] < met0->p[met0->np - 1])
00918
00919
         atm -> p[ip] = met0 -> p[met0 -> np - 1];
00920 }
00921
00923
00924 void module sedi(
00925
       ctl_t * ctl,
met_t * met0,
00926
00927
        met_t * met1,
00928
        atm_t * atm,
00929
        int ip,
00930
        double dt) {
00931
00932
       /\star Coefficients for Cunningham slip-flow correction (Kasten, 1968): \star/
       const double A = 1.249, B = 0.42, C = 0.87;
00933
00934
00935
        /\star Specific gas constant for dry air [J/(kg K)]: \star/
00936
       const double R = 287.058;
00937
00938
        /* Average mass of an air molecule [kg/molec]: */
00939
       const double m = 4.8096e-26;
00940
00941
        double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
00942
00943
        /* Check if parameters are available... */
00944
       if (ctl->qnt_r < 0 || ctl->qnt_rho < 0)</pre>
00945
          return;
00946
00947
       /* Convert units...
00948
       p = 100 * atm->p[ip];
        r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
00949
00950
       rho_p = atm->q[ctl->qnt_rho][ip];
00951
00952
        /* Get temperature... */
00953
       intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00954
                        atm->lat[ip], NULL, &T, NULL, NULL, NULL, NULL, NULL);
00955
       /* Density of dry air... */
rho = p / (R * T);
00956
00957
00958
       /* Dynamic viscosity of air... */ eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
00959
00960
00961
       /* Thermal velocity of an air molecule... */
v = sqrt(8 * GSL_CONST_MKSA_BOLTZMANN * T / (M_PI * m));
00962
00963
00964
00965
        /\star Mean free path of an air molecule... \star/
00966
       lambda = 2 * eta / (rho * v);
00967
00968
        /* Knudsen number for air... */
00969
        K = lambda / r_p;
00970
00971
        /* Cunningham slip-flow correction... */
00972
        G = 1 + K * (A + B * exp(-C / K));
00973
00974
        /* Sedimentation (fall) velocity... */
00975
00976
          2. * gsl_pow_2(r_p) * (rho_p
00977
                                 rho) * GSL_CONST_MKSA_GRAV_ACCEL / (9. * eta) * G;
00978
       /* Calculate pressure change... */ atm->p[ip] += dz2dp(v_p * dt / 1000., atm->p[ip]);
00979
00980
00981 }
00982
00984
00985 void write output (
00986
       const char *dirname,
        ctl_t * ctl,
met_t * met0,
00987
00988
00989
        met_t * met1,
00990
        atm_t * atm,
00991
       double t) {
00992
00993
       char filename[LEN];
00994
00995
00996
00997
       int year, mon, day, hour, min, sec;
00998
00999
       /* Get time... */
```

```
jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
          /* Write atmospheric data... */
if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
    sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_tab",
01002
01003
01004
                       dirname, ctl->atm_basename, year, mon, day, hour, min);
01005
01006
            write_atm(filename, ctl, atm, t);
01007
01008
01009
          /* Write CSI data... */
          if (ctl->csi_basename[0] != '-') {
01010
          sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01011
01012
            write_csi(filename, ctl, atm, t);
01013
01014
         /* Write ensemble data... */
if (ctl->ens_basename[0] != '-') {
   sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
   write_ens(filename, ctl, atm, t);
01015
01016
01017
01018
01019
01020
01021
          /* Write gridded data...
          if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01022
            sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d_tab",
01023
            dirname, ctl->grid_basename, year, mon, day, hour, min); write_grid(filename, ctl, met0, met1, atm, t);
01024
01025
01026
01027
01028
         /* Write profile data...
         if (ctl->prof_basename[0] != '-') {
01029
          sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01030
01031
            write_prof(filename, ctl, met0, met1, atm, t);
01032
01033
         /* Write station data... */
if (ctl->stat_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
    write station(filename ctl stm the ctl->stat_basename);
01034
01035
01036
            write_station(filename, ctl, atm, t);
01038
01039 }
```

## 5.33 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[])

# 5.33.1 Detailed Description

Create meteorological data files with synthetic wind fields.

Definition in file wind.c.

# 5.33.2 Function Documentation

5.33.2.1 void add\_text\_attribute ( int ncid, char \* varname, char \* attrname, char \* text )

Definition at line 173 of file wind.c.

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

Definition at line 41 of file wind.c.

```
00043
00044
00045
               ctl t ctl:
00046
               static char filename[LEN];
00047
00048
00049
               static double r, t0, z0, z1, dataLon[EX], dataLat[EY], dataZ[EP],
00050
                   u0, u1, alpha;
00051
               static float dataT[EP * EY * EX], dataU[EP * EY * EX], dataV[EP * EY * EX],
00052
00053
                   dataW[EP * EY * EX];
00054
00055
               static int ncid, dims[4], timid, levid, latid, lonid, tid, uid, vid, wid,
00056
                   idx, ix, iy, iz, nx, ny, nz, year, mon, day, hour, min, sec;
00057
00058
                /* Check arguments... */
00059
               if (argc < 3)
00060
                   ERRMSG("Give parameters: <ctl> <metbase>");
00061
00062
               /* Read control parameters... */
               read_ctl(argv[1], argc, argv, &ctl);
t0 = scan_ctl(argv[1], argc, argv, "WIND_TO", -1, "0", NULL);
00063
00064
               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);

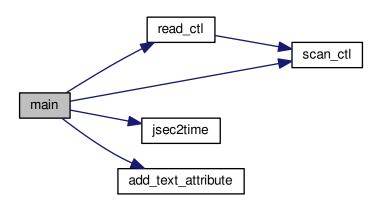
u1 = scan_ctl(argv[1], argc, argv, "WIND_UO", -1, "38.587660177302", NULL);

alpha = scan_ctl(argv[1], argc, argv, "WIND_LDHA", -1, "0.0", NULL);
00065
00066
00067
00068
00069
00070
00071
                alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);
00073
00074
                /* Check dimensions... */
00075
                if (nx < 1 || nx > EX)
00076
                   ERRMSG("Set 1 \leq NX \leq MAX!"):
                if (ny < 1 || ny > EY)
ERRMSG("Set 1 <= NY <= MAX!");
00077
00078
                if (nz < 1 || nz > EP)
00079
00080
                   ERRMSG("Set 1 <= NZ <= MAX!");</pre>
00081
00082
                /* Get time... */
                jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
t0 = year * 10000. + mon * 100. + day + hour / 24.;
00083
00084
00085
               /* Set filename... */
sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", argv[2], year, mon, day, hour);
00086
00087
00088
00089
                 /* Create netCDF file... */
00090
                NC(nc_create(filename, NC_CLOBBER, &ncid));
00091
               /* 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]));
00092
00093
00094
00095
00096
00097
00098
                /* Create variables...
               /* Create variables... */
NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));
NC(nc_def_var(ncid, "lev", NC_DOUBLE, 1, &dims[1], &levid));
NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[2], &latid));
NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[3], &lonid));
00099
00100
00101
00102
                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));
00103
00104
00105
                NC(nc_def_var(ncid, "W", NC_FLOAT, 4, &dims[0], &wid));
00106
00107
              /* Set attributes... */
add_text_attribute(ncid, "time", "long_name", "time");
add_text_attribute(ncid, "time", "units", "day as %Y$m%d.%f");
add_text_attribute(ncid, "lon", "long_name", "longitude");
add_text_attribute(ncid, "lon", "units", "degrees_east");
add_text_attribute(ncid, "lat", "long_name", "latitude");
add_text_attribute(ncid, "lat", "units", "degrees_north");
add_text_attribute(ncid, "lev", "long_name", "air_pressure");
add_text_attribute(ncid, "lev", "units", "Pa");
add_text_attribute(ncid, "T", "long_name", "Temperature");
add_text_attribute(ncid, "T", "units", "K");
add_text_attribute(ncid, "U", "long_name", "U velocity");
add_text_attribute(ncid, "V", "units", "m s**-1");
add_text_attribute(ncid, "V", "units", "m s**-1");
add_text_attribute(ncid, "V", "units", "m s**-1");
00108
                /* Set attributes... */
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
00119
00120
               add_text_attribute(ncid, "V", "units", "m s**-1");
```

5.34 wind.c 191

```
add_text_attribute(ncid, "W", "long_name", "Vertical velocity");
add_text_attribute(ncid, "W", "units", "Pa s**-1");
00124
00125
00126
         /* End definition... */
00127
        NC(nc_enddef(ncid));
00128
00129
         /* Set coordinates... */
00130
         for (ix = 0; ix < nx; ix++)
00131
          dataLon[ix] = 360.0 / nx * (double) ix;
         for (iy = 0; iy < ny; iy++)
  dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
for (iz = 0; iz < nz; iz++)</pre>
00132
00133
00134
          dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));
00135
00136
00137
         /* Write coordinates...
00138
         NC(nc_put_var_double(ncid, timid, &t0));
00139
         NC(nc_put_var_double(ncid, levid, dataZ));
         NC(nc_put_var_double(ncid, lonid, dataLon));
NC(nc_put_var_double(ncid, latid, dataLat));
00140
00141
00142
00143
         /* Create wind fields (Williamson et al., 1992)... */
00144
         for (ix = 0; ix < nx; ix++)
          for (iy = 0; iy < ny; iy++)</pre>
00145
             for (iz = 0; iz < nz; iz++) {
  idx = (iz * ny + iy) * nx + ix;</pre>
00146
00147
00148
                dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
00149
                                          * (cos(dataLat[iy] * M_PI / 180.0)
00150
                                             * cos(alpha * M_PI / 180.0)
                                             + sin(dataLat[iy] * M_PI / 180.0)
* cos(dataLon[ix] * M_PI / 180.0)
00151
00152
               00153
00154
00155
00156
                                         * sin(alpha * M_PI / 180.0));
00157
00158
00159
         /* Write wind data... */
         NC(nc_put_var_float(ncid, tid, dataT));
00160
00161
         NC(nc_put_var_float(ncid, uid, dataU));
00162
         NC(nc_put_var_float(ncid, vid, dataV));
00163
         NC(nc_put_var_float(ncid, wid, dataW));
00164
00165
         /* Close file...
00166
        NC (nc_close (ncid));
00168
         return EXIT_SUCCESS;
00169 }
```

Here is the call graph for this function:



### 5.34 wind.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
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
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
         Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /*
00028
          Functions...
00029
00030
00031 void add_text_attribute(
        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[EP * EY * EX], dataU[EP * EY * EX], dataV[EP * EY * EX],
00053
           dataW[EP * EY * EX];
00054
00055
         static int ncid, dims[4], timid, levid, latid, lonid, tid, uid, wid, wid,
00056
           idx, ix, iy, iz, nx, ny, nz, year, mon, day, hour, min, sec;
00057
00058
         /* Check arguments... */
00059
         if (argc < 3)
00060
           ERRMSG("Give parameters: <ctl> <metbase>");
00061
00062
         /* Read control parameters... */
         read_ctl(argv[1], argc, argv, &ctl);
00063
         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);
00064
00065
00066
00067
00068
00069
         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);
00070
00071
         alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);
00072
00073
00074
         /* Check dimensions... */
00075
         if (nx < 1 || nx > EX)
00076
           ERRMSG("Set 1 <= NX <= MAX!");</pre>
           f (ny < 1 || ny > EY)
ERRMSG("Set 1 <= NY <= MAX!");
00077
00078
         if (nz < 1 || nz > EP)
   ERRMSG("Set 1 <= NZ <= MAX!");</pre>
00079
00080
00081
         /* Get time... */
00082
00083
          jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
00084
         t0 = year * 10000. + mon * 100. + day + hour / 24.;
00085
         /* Set filename... */
sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", argv[2], year, mon, day, hour);
00086
00087
00088
          /* Create netCDF file...
00089
00090
         NC(nc_create(filename, NC_CLOBBER, &ncid));
00091
         /* Create dimensions... */
NC(nc_def_dim(ncid, "time", 1, &dims[0]));
00092
00093
```

5.34 wind.c 193

```
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]));
00095
00096
00097
           /* 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));
00098
00099
00100
00101
           NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[3], &lonid));
00102
           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));
00103
00104
00105
00106
           NC(nc_def_var(ncid, "W", NC_FLOAT, 4, &dims[0], &wid));
00107
00108
            /* Set attributes... */
           /* 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", "units", "Fa");
add_text_attribute(ncid, "T", "units", "K");
add_text_attribute(ncid, "T", "units", "K");
add_text_attribute(ncid, "T", "units", "R");
add_text_attribute(ncid, "T", "units", "R");
add_text_attribute(ncid, "T", "units", "R");
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
           add_text_attribute(ncid, "U", "long_name", "U velocity");
add_text_attribute(ncid, "U", "units", "m s**-1");
00119
00120
            add_text_attribute(ncid, "V", "long_name", "V velocity");
00121
           add_text_attribute(ncid, "V", "units", "m s**-1");
add_text_attribute(ncid, "W", "long_name", "Vertical velocity");
add_text_attribute(ncid, "W", "long_name", "Vertical velocity");
00122
00123
00124
00125
00126
            /* End definition... */
00127
           NC(nc_enddef(ncid));
00128
            /* Set coordinates... */
00129
           for (ix = 0; ix < nx; ix++)
00130
              dataLon[ix] = 360.0 / nx * (double) ix;
00132
           for (iy = 0; iy < ny; iy++)
00133
              dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
00134
            for (iz = 0; iz < nz; iz++)
              dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));
00135
00136
00137
            /* Write coordinates... */
00138
           NC(nc_put_var_double(ncid, timid, &t0));
00139
           NC(nc_put_var_double(ncid, levid, dataZ));
00140
           NC(nc_put_var_double(ncid, lonid, dataLon));
00141
           NC(nc_put_var_double(ncid, latid, dataLat));
00142
00143
            /* Create wind fields (Williamson et al., 1992)... */
00144
           for (ix = 0; ix < nx; ix++)
00145
              for (iy = 0; iy < ny; iy++)</pre>
00146
                  for (iz = 0; iz < nz; iz++) {
                     idx = (iz * ny + iy) * nx + ix;
00147
                     dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
00148
                                                      * (cos(dataLat[iy] * M_PI / 180.0)

* cos(alpha * M_PI / 180.0)
00149
                                                           + sin(dataLat[iy] * M_PI / 180.0)
00151
00152
                                                           * cos(dataLon[ix] * M_PI / 180.0)
                                                           * sin(alpha * M_PI / 180.0)));
00153
                    00154
00155
00156
                                                      * sin(alpha * M_PI / 180.0));
00157
00158
00159
            /* Write wind data... */
00160
           NC(nc_put_var_float(ncid, tid, dataT));
00161
           NC(nc_put_var_float(ncid, uid, dataU));
00162
           NC(nc_put_var_float(ncid, vid, dataV));
00163
           NC(nc_put_var_float(ncid, wid, dataW));
00164
00165
           /* Close file... */
00166
           NC(nc_close(ncid));
00167
00168
           return EXIT SUCCESS;
00170
00172
00173 void add_text_attribute(
00174 int ncid,
           char *varname,
           char *attrname,
00176
00177
           char *text) {
00178
00179
           int varid;
00180
```

```
00181     NC(nc_inq_varid(ncid, varname, &varid));
00182     NC(nc_put_att_text(ncid, varid, attrname, strlen(text), text));
00183 }
```

# Index

add_text_attribute	ctl_t, 14
wind.c, 189	ctl_t, 5
atm_basename	atm_basename, 13
ctl_t, 13	atm_dt_out, 13
atm_dt_out	atm_filter, 13
ctl_t, 13	atm_gpfile, 13
atm_filter	balloon, 12
ctl_t, 13	csi_basename, 13
atm_gpfile	csi_dt_out, 14
ctl_t, 13	csi_lat0, 15
atm_t, 3	csi_lat1, 15
lat, 4	csi_lon0, 14
lon, 4	csi_lon1, 15
np, 4	csi_modmin, 14
p, 4 g, 4	csi_nx, 14
time, 4	csi_ny, 15
up, 4	csi_nz, 14
vp, 4	csi_obsfile, 14
wp, 5	csi_obsmin, 14
wp, 0	csi_z0, 14
balloon	csi_z1, 14
ctl_t, 12	direction, 11
<del>-</del> '	dt_met, 11
cart2geo	dt_mod, 11
libtrac.c, 41	ens_basename, 18
libtrac.h, 96	grid_basename, 15
center.c, 21	grid_dt_out, 15
main, 22	grid_gpfile, 15
csi_basename	grid_lat0, 16
ctl_t, 13	grid_lat1, 16
csi_dt_out	grid_lon0, 16
ctl_t, 14	grid_lon1, 16
csi_lat0	grid_nx, 16
ctl_t, 15	grid_ny, 16 grid_nz, 15
csi_lat1	<b>-</b>
ctl_t, 15	grid_sparse, 15
csi_lon0	grid_z0, 16
ctl_t, 14	grid_z1, 16 isosurf, 12
csi_lon1	met_np, 12
ctl_t, 15	met_np, 12
csi_modmin	nq, 9
ctl_t, 14	prof_basename, 16
csi_nx	prof lat0, 17
ctl_t, 14	prof lat1, 18
csi_ny	prof_lon0, 17
ctl_t, 15 csi nz	prof_lon1, 17
ctl t, 14	prof_nx, 17
csi obsfile	prof_ny, 17
ctl_t, 14	prof_nz, 17
csi obsmin	prof obsfile, 17
ctl_t, 14	prof_z0, 17
csi z0	prof_z1, 17
ctl_t, 14	psc h2o, 13
csi_z1	psc_hno3, 13
	F-1555, 10

qnt_ens, 9	libtrac.c, 42
qnt_format, 9	libtrac.h, 97
qnt_h2o, 10	
qnt_m, 9	ens_basename
qnt_name, 9	ctl_t, 18
qnt_o3, 10	extract.c, 32
qnt_p, 10	main, <mark>32</mark>
qnt_ps, 9	
qnt_pv, 10	geo2cart
qnt_r, 9	libtrac.c, 42
qnt_rho, 9	libtrac.h, 97
qnt_stat, 11	get_met
qnt_t, 10	libtrac.c, 43
qnt_theta, 10	libtrac.h, 97
qnt_tice, 11	get_met_help
qnt_tnat, 11	libtrac.c, 44
qnt_tsts, 11	libtrac.h, 98
qnt_u, 10	grid_basename
qnt_unit, 9	ctl_t, 15
qnt_v, 10	grid_dt_out
qnt_w, 10	ctl_t, 15
stat_basename, 18	grid_gpfile
stat_lat, 18	ctl_t, 15
stat_lon, 18	grid_lat0
stat_r, 18	ctl_t, 16
t_start, 11	grid_lat1
t_stop, 11	ctl_t, 16
tdec_strat, 13	grid_lon0
tdec_trop, 13	ctl_t, 16
turb_dx_strat, 12	grid_lon1
turb_dx_trop, 12	ctl_t, 16
turb_dz_strat, 12	grid_nx
turb_dz_trop, 12	ctl_t, 16
turb_meso, 12	grid_ny
de vOde	ctl_t, 16
deg2dx	grid_nz
libtrac.c, 41	ctl_t, 15
libtrac.h, 96	grid_sparse
deg2dy libtrac.c, 41	ctl_t, 15
libtrac.h, 96	grid_z0
direction	ctl_t, 16
ctl_t, 11	grid_z1
dist.c, 25	ctl_t, 16
main, 26	h2o
dp2dz	met t, 21
libtrac.c, 42	1110t_t, <b>2</b> 1
libtrac.h, 96	init.c, 34
dt met	main, <b>35</b>
ctl_t, 11	init_simtime
dt mod	trac.c, 163
ctl_t, 11	intpol_met_2d
dx2deg	libtrac.c, 44
libtrac.c, 42	libtrac.h, 99
libtrac.h, 96	intpol_met_3d
dy2deg	libtrac.c, 45
libtrac.c, 42	libtrac.h, 99
libtrac.h, 97	intpol_met_space
dz2dp	libtrac.c, 45

libtrac.h, 100	get_met, 97
intpol_met_time	get_met_help, 98
libtrac.c, 46	intpol_met_2d, 99
libtrac.h, 101	intpol_met_3d, 99
isosurf	intpol_met_space, 100
ctl_t, 12	intpol_met_time, 101
	jsec2time, 102
jsec2time	locate, 102
libtrac.c, 47	read_atm, 103
libtrac.h, 102	read_ctl, 103
jsec2time.c, 38	read_met, 106
main, 38	read_met_extrapolate, 108
	read_met_help, 109
lat	read_met_ml2pl, 109
atm_t, 4	read_met_periodic, 110
met_t, 20	scan_ctl, 110
libtrac.c, 39	time2jsec, 111
cart2geo, 41	timer, 112
deg2dx, 41	tropopause, 112
deg2dy, 41	write_atm, 114
dp2dz, 42	write_csi, 115
dx2deg, 42	write_ens, 117
dy2deg, 42	write_grid, 119
dz2dp, 42	write_prof, 121
geo2cart, 42	write_station, 124
get_met, 43	locate
get_met_help, 44	libtrac.c, 47
intpol_met_2d, 44	libtrac.h, 102
intpol_met_3d, 45	lon
intpol_met_space, 45	
intpoi_met_space, 45	atm t. 4
intpol_met_time, 46	atm_t, 4 met_t, 20
· ·	atm_t, 4 met_t, 20
intpol_met_time, 46	<del>-</del> :
intpol_met_time, 46 jsec2time, 47	met_t, 20
intpol_met_time, 46 jsec2time, 47 locate, 47	met_t, 20
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48	met_t, 20 main center.c, 22
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48	met_t, 20 main center.c, 22 dist.c, 26
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52	met_t, 20 main center.c, 22 dist.c, 26 extract.c, 32
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54	met_t, 20 main center.c, 22 dist.c, 26 extract.c, 32 init.c, 35
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54	met_t, 20 main center.c, 22 dist.c, 26 extract.c, 32 init.c, 35 jsec2time.c, 38
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54 read_met_ml2pl, 55	met_t, 20  main center.c, 22 dist.c, 26 extract.c, 32 init.c, 35 jsec2time.c, 38 match.c, 132
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54 read_met_ml2pl, 55 read_met_periodic, 55	met_t, 20  main  center.c, 22 dist.c, 26 extract.c, 32 init.c, 35 jsec2time.c, 38 match.c, 132 met_map.c, 137
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54 read_met_ml2pl, 55 read_met_periodic, 55 scan_ctl, 56	met_t, 20  main  center.c, 22 dist.c, 26 extract.c, 32 init.c, 35 jsec2time.c, 38 match.c, 132 met_map.c, 137 met_prof.c, 141
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54 read_met_ml2pl, 55 read_met_periodic, 55 scan_ctl, 56 time2jsec, 57	met_t, 20  main  center.c, 22  dist.c, 26  extract.c, 32  init.c, 35  jsec2time.c, 38  match.c, 132  met_map.c, 137  met_prof.c, 141  met_sample.c, 145
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54 read_met_ml2pl, 55 read_met_periodic, 55 scan_ctl, 56 time2jsec, 57 timer, 57	met_t, 20  main  center.c, 22 dist.c, 26 extract.c, 32 init.c, 35 jsec2time.c, 38 match.c, 132 met_map.c, 137 met_prof.c, 141 met_sample.c, 145 met_zm.c, 149
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54 read_met_ml2pl, 55 read_met_periodic, 55 scan_ctl, 56 time2jsec, 57 timer, 57 tropopause, 58	met_t, 20  main  center.c, 22 dist.c, 26 extract.c, 32 init.c, 35 jsec2time.c, 38 match.c, 132 met_map.c, 137 met_prof.c, 141 met_sample.c, 145 met_zm.c, 149 smago.c, 153
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54 read_met_ml2pl, 55 read_met_periodic, 55 scan_ctl, 56 time2jsec, 57 timer, 57 tropopause, 58 write_atm, 60	met_t, 20  main  center.c, 22 dist.c, 26 extract.c, 32 init.c, 35 jsec2time.c, 38 match.c, 132 met_map.c, 137 met_prof.c, 141 met_sample.c, 145 met_zm.c, 149 smago.c, 153 split.c, 157
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54 read_met_ml2pl, 55 read_met_periodic, 55 scan_ctl, 56 time2jsec, 57 timer, 57 tropopause, 58 write_atm, 60 write_csi, 61	met_t, 20  main  center.c, 22 dist.c, 26 extract.c, 32 init.c, 35 jsec2time.c, 38 match.c, 132 met_map.c, 137 met_prof.c, 141 met_sample.c, 145 met_zm.c, 149 smago.c, 153 split.c, 157 time2jsec.c, 161
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54 read_met_ml2pl, 55 read_met_periodic, 55 scan_ctl, 56 time2jsec, 57 timer, 57 tropopause, 58 write_atm, 60 write_csi, 61 write_ens, 63	met_t, 20  main  center.c, 22 dist.c, 26 extract.c, 32 init.c, 35 jsec2time.c, 38 match.c, 132 met_map.c, 137 met_prof.c, 141 met_sample.c, 145 met_zm.c, 149 smago.c, 153 split.c, 157 time2jsec.c, 161 trac.c, 173
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54 read_met_ml2pl, 55 read_met_periodic, 55 scan_ctl, 56 time2jsec, 57 timer, 57 tropopause, 58 write_atm, 60 write_csi, 61 write_ens, 63 write_grid, 65	met_t, 20  main  center.c, 22 dist.c, 26 extract.c, 32 init.c, 35 jsec2time.c, 38 match.c, 132 met_map.c, 137 met_prof.c, 141 met_sample.c, 145 met_zm.c, 149 smago.c, 153 split.c, 157 time2jsec.c, 161 trac.c, 173 wind.c, 189
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54 read_met_ml2pl, 55 read_met_periodic, 55 scan_ctl, 56 time2jsec, 57 timer, 57 tropopause, 58 write_atm, 60 write_csi, 61 write_prof, 65 write_prof, 67	met_t, 20  main  center.c, 22 dist.c, 26 extract.c, 32 init.c, 35 jsec2time.c, 38 match.c, 132 met_map.c, 137 met_prof.c, 141 met_sample.c, 145 met_zm.c, 149 smago.c, 153 split.c, 157 time2jsec.c, 161 trac.c, 173 wind.c, 189 match.c, 132
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54 read_met_periodic, 55 read_met_periodic, 55 scan_ctl, 56 time2jsec, 57 timer, 57 tropopause, 58 write_atm, 60 write_csi, 61 write_ens, 63 write_prof, 67 write_station, 69 libtrac.h, 94	met_t, 20  main  center.c, 22 dist.c, 26 extract.c, 32 init.c, 35 jsec2time.c, 38 match.c, 132 met_map.c, 137 met_prof.c, 141 met_sample.c, 145 met_zm.c, 149 smago.c, 153 split.c, 157 time2jsec.c, 161 trac.c, 173 wind.c, 189  match.c, 132 main, 132 met_map.c, 137
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54 read_met_periodic, 55 read_met_periodic, 55 scan_ctl, 56 time2jsec, 57 timer, 57 tropopause, 58 write_atm, 60 write_csi, 61 write_ens, 63 write_grid, 65 write_station, 69	met_t, 20  main  center.c, 22 dist.c, 26 extract.c, 32 init.c, 35 jsec2time.c, 38 match.c, 132 met_map.c, 137 met_prof.c, 141 met_sample.c, 145 met_zm.c, 149 smago.c, 153 split.c, 157 time2jsec.c, 161 trac.c, 173 wind.c, 189 match.c, 132 main, 132
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54 read_met_periodic, 55 read_met_periodic, 55 scan_ctl, 56 time2jsec, 57 timer, 57 tropopause, 58 write_atm, 60 write_csi, 61 write_ens, 63 write_grid, 65 write_station, 69 libtrac.h, 94 cart2geo, 96 deg2dx, 96	met_t, 20  main
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54 read_met_ml2pl, 55 read_met_periodic, 55 scan_ctl, 56 time2jsec, 57 timer, 57 tropopause, 58 write_atm, 60 write_csi, 61 write_ens, 63 write_grid, 65 write_prof, 67 write_station, 69 libtrac.h, 94 cart2geo, 96 deg2dx, 96 deg2dy, 96	met_t, 20  main
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54 read_met_ml2pl, 55 read_met_periodic, 55 scan_ctl, 56 time2jsec, 57 timer, 57 tropopause, 58 write_atm, 60 write_csi, 61 write_ens, 63 write_grid, 65 write_prof, 67 write_station, 69 libtrac.h, 94 cart2geo, 96 deg2dx, 96 deg2dy, 96 dp2dz, 96	met_t, 20  main
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54 read_met_ml2pl, 55 read_met_periodic, 55 scan_ctl, 56 time2jsec, 57 timer, 57 tropopause, 58 write_atm, 60 write_csi, 61 write_ens, 63 write_grid, 65 write_prof, 67 write_station, 69 libtrac.h, 94 cart2geo, 96 deg2dx, 96 deg2dy, 96 dp2dz, 96 dx2deg, 96	met_t, 20  main  center.c, 22 dist.c, 26 extract.c, 32 init.c, 35 jsec2time.c, 38 match.c, 132 met_map.c, 137 met_prof.c, 141 met_sample.c, 145 met_zm.c, 149 smago.c, 153 split.c, 157 time2jsec.c, 161 trac.c, 173 wind.c, 189  match.c, 132 main, 132 met_map.c, 137 main, 137  met_np ctl_t, 12  met_p ctl_t, 12
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54 read_met_ml2pl, 55 read_met_periodic, 55 scan_ctl, 56 time2jsec, 57 timer, 57 tropopause, 58 write_atm, 60 write_csi, 61 write_ens, 63 write_grid, 65 write_prof, 67 write_station, 69 libtrac.h, 94 cart2geo, 96 deg2dx, 96 deg2dy, 96 dp2dz, 96 dx2deg, 96 dy2deg, 97	met_t, 20  main
intpol_met_time, 46 jsec2time, 47 locate, 47 read_atm, 48 read_ctl, 48 read_met, 52 read_met_extrapolate, 54 read_met_help, 54 read_met_ml2pl, 55 read_met_periodic, 55 scan_ctl, 56 time2jsec, 57 timer, 57 tropopause, 58 write_atm, 60 write_csi, 61 write_ens, 63 write_grid, 65 write_prof, 67 write_station, 69 libtrac.h, 94 cart2geo, 96 deg2dx, 96 deg2dy, 96 dp2dz, 96 dx2deg, 96	met_t, 20  main  center.c, 22 dist.c, 26 extract.c, 32 init.c, 35 jsec2time.c, 38 match.c, 132 met_map.c, 137 met_prof.c, 141 met_sample.c, 145 met_zm.c, 149 smago.c, 153 split.c, 157 time2jsec.c, 161 trac.c, 173 wind.c, 189  match.c, 132 main, 132 met_map.c, 137 main, 137  met_np ctl_t, 12  met_p ctl_t, 12

main, 145	ctl_t, 18
met_t, 18	prof_lon0
h2o, 21	ctl_t, 17
lat, 20	prof_lon1
lon, 20	ctl_t, 17
np, 20	prof_nx
nx, 19	ctl_t, 17
ny, 20	prof_ny
o3, <mark>21</mark>	ctl_t, 17
p, 20	prof_nz
pl, 20	ctl_t, 17
ps, 20	prof_obsfile
t, 20	ctl_t, 17
time, 19	prof_z0
u, 20	ctl_t, 17
v, 21	prof_z1
w, 21	ctl_t, 17
met_zm.c, 148	ps
main, 149	met_t, 20
module_advection	psc_h2o
trac.c, 163	ctl_t, 13
module_decay	psc_hno3
trac.c, 163	ctl_t, 13
module_diffusion_meso	q
trac.c, 164	atm_t, 4
module_diffusion_turb	qnt_ens
trac.c, 166	ctl_t, 9
module_isosurf	qnt_format
trac.c, 167	ctl_t, 9
module_meteo	qnt_h2o
trac.c, 169	ctl_t, 10
module_position	qnt_m
trac.c, 170	ctl_t, 9
module_sedi	qnt_name
trac.c, 171	ctl_t, 9
nn	qnt_o3
np	ctl_t, 10
atm_t, 4 met_t, 20	qnt_p
	ctl_t, 10
nq ctl t Q	qnt_ps
ctl_t, 9	ctl_t, 9
nx met_t, 19	qnt_pv
	ctl_t, 10
ny met t 20	qnt_r
met_t, 20	ctl_t, 9
03	qnt_rho
met_t, 21	ctl_t, 9
11101_1, 21	qnt_stat
p	ctl_t, 11
atm_t, 4	qnt_t
met_t, 20	ctl_t, 10
pl	qnt_theta
met_t, 20	ctl_t, 10
prof_basename	qnt_tice
ctl_t, 16	ctl_t, 11
prof lat0	qnt tnat
ctl_t, 17	ctl_t, 11
prof_lat1	qnt tsts
r - = m-	4

ctl_t, 11	atm_t, 4
qnt_u	met_t, 19
ctl_t, 10	time2jsec
qnt_unit	libtrac.c, 57
ctl_t, 9	libtrac.h, 111
qnt_v	time2jsec.c, 160
ctl_t, 10	main, 161
qnt_w	timer
ctl_t, 10	libtrac.c, 57
	libtrac.h, 112
read_atm	trac.c, 162
libtrac.c, 48	init_simtime, 163
libtrac.h, 103	main, 173
read_ctl	module advection, 163
libtrac.c, 48	module_decay, 163
libtrac.h, 103	module diffusion meso, 164
read met	module_diffusion_turb, 166
libtrac.c, 52	module_isosurf, 167
libtrac.h, 106	module_meteo, 169
read_met_extrapolate	module_position, 170
libtrac.c, 54	module sedi, 171
libtrac.h, 108	write_output, 172
read_met_help	tropopause
libtrac.c, 54	libtrac.c, 58
libtrac.h, 109	libtrac.h, 112
read_met_ml2pl	turb_dx_strat
libtrac.c, 55	
libtrac.h, 109	ctl_t, 12
read_met_periodic	turb_dx_trop
libtrac.c, 55	ctl_t, 12
libtrac.h, 110	turb_dz_strat
iibiiac.ii, 110	ctl_t, 12
scan ctl	turb_dz_trop
libtrac.c, 56	ctl_t, 12
libtrac.h, 110	turb_meso
smago.c, 153	ctl_t, 12
<del>-</del>	
main, 153	u
split.c, 156	met_t, 20
main, 157	up
stat_basename	atm_t, 4
ctl_t, 18	
stat_lat	V
ctl_t, 18	met_t, 21
stat_lon	vp
ctl_t, 18	atm_t, 4
stat_r	
ctl_t, 18	W
	met_t, 21
t	wind.c, 189
met_t, 20	add_text_attribute, 189
t_start	main, 189
ctl_t, 11	wp
t_stop	atm_t, 5
ctl_t, 11	write_atm
tdec_strat	libtrac.c, 60
ctl_t, 13	libtrac.h, 114
tdec_trop	write_csi
ctl_t, 13	libtrac.c, 61
time	libtrac.h, 115

write\_ens
libtrac.c, 63
libtrac.h, 117
write\_grid
libtrac.c, 65
libtrac.h, 119
write\_output
trac.c, 172
write\_prof
libtrac.c, 67
libtrac.h, 121
write\_station
libtrac.c, 69
libtrac.h, 124