MPTRAC

Generated by Doxygen 1.8.11

ii CONTENTS

Contents

1	Main	Page		2
2	Data	Struct	ure Index	2
	2.1	Data S	Structures	2
3	File	Index		2
	3.1	File Lis	st	2
4	Data	Struct	ure 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 St	truct Reference	5
		4.2.1	Detailed Description	8
		4.2.2	Field Documentation	8
	4.3	met t	Struct Reference	17
		4.3.1	Detailed Description	18
		4.3.2	Field Documentation	18
5	File	Docum	entation	19
	5.1	center.	.c File Reference	19
		5.1.1	Detailed Description	20
		5.1.2	Function Documentation	20
	5.2	center.	.c	22
	5.3	dist.c F	File Reference	24
		5.3.1	Detailed Description	24
		5.3.2	Function Documentation	24
	5.4	dist.c .		27
	5.5	extract	t.c File Reference	30
		5.5.1	Detailed Description	30
		5.5.2	Function Documentation	30

5.6	extract.c	31
5.7	init.c File Reference	32
	5.7.1 Detailed Description	33
	5.7.2 Function Documentation	33
5.8	init.c	34
5.9	jsec2time.c File Reference	36
	5.9.1 Detailed Description	36
	5.9.2 Function Documentation	36
5.10	jsec2time.c	37
5.11	libtrac.c File Reference	37
	5.11.1 Detailed Description	39
	5.11.2 Function Documentation	39
5.12	libtrac.c	65
5.13	libtrac.h File Reference	85
	5.13.1 Detailed Description	87
	5.13.2 Function Documentation	87
5.14	libtrac.h	113
5.15	match.c File Reference	119
	5.15.1 Detailed Description	119
	5.15.2 Function Documentation	119
5.16	match.c	122
5.17	met_map.c File Reference	124
	5.17.1 Detailed Description	124
	5.17.2 Function Documentation	124
5.18	met_map.c	126
5.19	met_prof.c File Reference	128
	5.19.1 Detailed Description	128
	5.19.2 Function Documentation	128
5.20	met_prof.c	130
5.21	met_sample.c File Reference	132

	5.21.1 Detailed Description	133
	5.21.2 Function Documentation	133
5.22	2 met_sample.c	134
5.23	B met_zm.c File Reference	135
	5.23.1 Detailed Description	136
	5.23.2 Function Documentation	136
5.24	met_zm.c	138
5.25	5 smago.c File Reference	140
	5.25.1 Detailed Description	140
	5.25.2 Function Documentation	140
5.26	S smago.c	142
5.27	7 split.c File Reference	143
	5.27.1 Detailed Description	143
	5.27.2 Function Documentation	144
5.28	3 split.c	146
5.29	time2jsec.c File Reference	147
	5.29.1 Detailed Description	147
	5.29.2 Function Documentation	148
5.30) time2jsec.c	148
5.31	trac.c File Reference	149
	5.31.1 Detailed Description	149
	5.31.2 Function Documentation	150
5.32	2 trac.c	163
5.33	3 wind.c File Reference	174
	5.33.1 Detailed Description	175
	5.33.1 Detailed Description	
5 2/	5.33.1 Detailed Description	175

181

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	17

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

center.c Calculate center of mass of air parcels	19
dist.c Calculate transport deviations of trajectories	24
extract.c Extract single trajectory from atmospheric data files	30
init.c Create atmospheric data file with initial air parcel positions	32
jsec2time.c Convert Julian seconds to date	36
libtrac.c MPTRAC library definitions	37
libtrac.h MPTRAC library declarations	85
match.c Calculate deviations between two trajectories	119

met_map.c Extract global map from meteorological data	124
met_prof.c Extract vertical profile from meteorological data	128
met_sample.c Sample meteorological data at given geolocations	132
met_zm.c Extract zonal mean from meteorological data	
smago.c Estimate horizontal diffusivity based on Smagorinsky theory	140
split.c Split air parcels into a larger number of parcels	143
time2jsec.c Convert date to Julian seconds	147
trac.c Lagrangian particle dispersion model	149
wind.c Create meteorological data files with synthetic wind fields	174
4 Data Structure Documentation	
4.1 atm_t Struct Reference	
Atmospheric data.	
<pre>#include <libtrac.h></libtrac.h></pre>	
Data Fields	
 int np Number of air pacels. double time [NP] Time [s]. double p [NP] Pressure [hPa]. double lon [NP] Longitude [deg]. double lat [NP] Latitude [deg]. double q [NQ][NP] Quantitiy data (for various, user-defined attributes). double up [NP] Zonal wind perturbation [m/s]. double vp [NP] 	
Meridional wind perturbation [m/s]. • double wp [NP]	

Vertical velocity perturbation [hPa/s].

```
4.1.1 Detailed Description
Atmospheric data.
Definition at line 404 of file libtrac.h.
4.1.2 Field Documentation
4.1.2.1 int atm_t::np
Number of air pacels.
Definition at line 407 of file libtrac.h.
4.1.2.2 double atm_t::time[NP]
Time [s].
Definition at line 410 of file libtrac.h.
4.1.2.3 double atm_t::p[NP]
Pressure [hPa].
Definition at line 413 of file libtrac.h.
4.1.2.4 double atm_t::lon[NP]
Longitude [deg].
Definition at line 416 of file libtrac.h.
4.1.2.5 double atm_t::lat[NP]
Latitude [deg].
Definition at line 419 of file libtrac.h.
4.1.2.6 double atm_t::q[NQ][NP]
Quantitiy data (for various, user-defined attributes).
Definition at line 422 of file libtrac.h.
4.1.2.7 double atm_t::up[NP]
Zonal wind perturbation [m/s].
Definition at line 425 of file libtrac.h.
```

4.1.2.8 double atm_t::vp[NP] Meridional wind perturbation [m/s]. Definition at line 428 of file libtrac.h. 4.1.2.9 double atm_t::wp[NP] Vertical velocity perturbation [hPa/s]. Definition at line 431 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_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_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_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 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) $[m^2/2]$.

· double turb dx strat

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

double turb_dz_trop

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

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].

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].

• char csi_basename [LEN]

Basename of CSI data files.

· double csi_dt_out

Time step for CSI data output [s].

• char csi_obsfile [LEN]

Observation data file for CSI analysis.

• double csi_obsmin

Minimum observation index to trigger detection.

• double csi_modmin

Minimum column density to trigger detection [kg/m^{\(\)}2].

• int csi_nz

Number of altitudes of gridded CSI data.

• double csi_z0

Lower altitude of gridded CSI data [km].

• double csi z1

Upper altitude of gridded CSI data [km].

• int csi nx

Number of longitudes of gridded CSI data.

double csi lon0

Lower longitude of gridded CSI data [deg].

double csi_lon1

Upper longitude of gridded CSI data [deg].

int csi_ny

Number of latitudes of gridded CSI data.

• double csi_lat0

Lower latitude of gridded CSI data [deg].

double csi_lat1

Upper latitude of gridded CSI data [deg].

• char grid_basename [LEN]

Basename of grid data files.

char grid_gpfile [LEN]

Gnuplot file for gridded data.

· double grid_dt_out

Time step for gridded data output [s].

int grid_sparse

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

• int grid_nz

Number of altitudes of gridded data.

double grid_z0

Lower altitude of gridded data [km].

• double grid z1

Upper altitude of gridded data [km].

int grid_nx

Number of longitudes of gridded data.

• double grid_lon0

Lower longitude of gridded data [deg].

double grid_lon1

Upper longitude of gridded data [deg].

• int grid_ny

Number of latitudes of gridded data.

double grid_lat0

Lower latitude of gridded data [deg].

• double grid_lat1

Upper latitude of gridded data [deg].

char prof_basename [LEN]

Basename for profile output file.

char prof_obsfile [LEN]

Observation data file for profile output.

• int prof_nz

Number of altitudes of gridded profile data.

double prof_z0

Lower altitude of gridded profile data [km].

```
double prof_z1
           Upper altitude of gridded profile data [km].
     • int prof_nx
           Number of longitudes of gridded profile data.

    double prof_lon0

           Lower longitude of gridded profile data [deg].

    double prof_lon1

           Upper longitude of gridded profile data [deg].
     • int prof_ny
           Number of latitudes of gridded profile data.

    double prof_lat0

           Lower latitude of gridded profile data [deg].

    double prof_lat1

           Upper latitude of gridded profile data [deg].

    char 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 173 of file libtrac.h.
4.2.2 Field Documentation
4.2.2.1 int ctl_t::nq
Number of quantities.
Definition at line 176 of file libtrac.h.
4.2.2.2 char ctl_t::qnt_name[NQ][LEN]
Quantity names.
Definition at line 179 of file libtrac.h.
4.2.2.3 char ctl_t::qnt_unit[NQ][LEN]
Quantity units.
Definition at line 182 of file libtrac.h.
```

4.2.2.4 char ctl_t::qnt_format[NQ][LEN] Quantity output format. Definition at line 185 of file libtrac.h. 4.2.2.5 int ctl_t::qnt_m Quantity array index for mass. Definition at line 188 of file libtrac.h. 4.2.2.6 int ctl_t::qnt_rho Quantity array index for particle density. Definition at line 191 of file libtrac.h. 4.2.2.7 int ctl_t::qnt_r Quantity array index for particle radius. Definition at line 194 of file libtrac.h. 4.2.2.8 int ctl_t::qnt_ps Quantity array index for surface pressure. Definition at line 197 of file libtrac.h. 4.2.2.9 int ctl_t::qnt_t Quantity array index for temperature. Definition at line 200 of file libtrac.h. 4.2.2.10 int ctl_t::qnt_u Quantity array index for zonal wind. Definition at line 203 of file libtrac.h. 4.2.2.11 int ctl_t::qnt_v Quantity array index for meridional wind. Definition at line 206 of file libtrac.h. 4.2.2.12 int ctl_t::qnt_w Quantity array index for vertical velocity.

Definition at line 209 of file libtrac.h.

```
4.2.2.13 int ctl_t::qnt_h2o
Quantity array index for water vapor vmr.
Definition at line 212 of file libtrac.h.
4.2.2.14 int ctl_t::qnt_o3
Quantity array index for ozone vmr.
Definition at line 215 of file libtrac.h.
4.2.2.15 int ctl_t::qnt_theta
Quantity array index for potential temperature.
Definition at line 218 of file libtrac.h.
4.2.2.16 int ctl_t::qnt_stat
Quantity array index for station flag.
Definition at line 221 of file libtrac.h.
4.2.2.17 int ctl_t::direction
Direction flag (1=forward calculation, -1=backward calculation).
Definition at line 224 of file libtrac.h.
4.2.2.18 double ctl_t::t_start
Start time of simulation [s].
Definition at line 227 of file libtrac.h.
4.2.2.19 double ctl_t::t_stop
Stop time of simulation [s].
Definition at line 230 of file libtrac.h.
4.2.2.20 double ctl_t::dt_mod
Time step of simulation [s].
Definition at line 233 of file libtrac.h.
4.2.2.21 double ctl_t::dt_met
Time step of meteorological data [s].
Definition at line 236 of file libtrac.h.
```

```
4.2.2.22 int ctl_t::isosurf
Isosurface parameter (0=none, 1=pressure, 2=density, 3=theta, 4=balloon).
Definition at line 240 of file libtrac.h.
4.2.2.23 char ctl_t::balloon[LEN]
Balloon position filename.
Definition at line 243 of file libtrac.h.
4.2.2.24 double ctl_t::turb_dx_trop
Horizontal turbulent diffusion coefficient (troposphere) [m^2/s].
Definition at line 246 of file libtrac.h.
4.2.2.25 double ctl_t::turb_dx_strat
Horizontal turbulent diffusion coefficient (stratosphere) [m<sup>2</sup>/s].
Definition at line 249 of file libtrac.h.
4.2.2.26 double ctl_t::turb_dz_trop
Vertical turbulent diffusion coefficient (troposphere) [m<sup>2</sup>/s].
Definition at line 252 of file libtrac.h.
4.2.2.27 double ctl_t::turb_dz_strat
Vertical turbulent diffusion coefficient (stratosphere) [m<sup>2</sup>/s].
Definition at line 255 of file libtrac.h.
4.2.2.28 double ctl_t::turb_meso
Scaling factor for mesoscale wind fluctuations.
Definition at line 258 of file libtrac.h.
4.2.2.29 double ctl_t::tdec_trop
Life time of particles (troposphere) [s].
Definition at line 261 of file libtrac.h.
4.2.2.30 double ctl_t::tdec_strat
Life time of particles (stratosphere) [s].
Definition at line 264 of file libtrac.h.
```

```
4.2.2.31 char ctl_t::atm_basename[LEN]
Basename of atmospheric data files.
Definition at line 267 of file libtrac.h.
4.2.2.32 char ctl_t::atm_gpfile[LEN]
Gnuplot file for atmospheric data.
Definition at line 270 of file libtrac.h.
4.2.2.33 double ctl_t::atm_dt_out
Time step for atmospheric data output [s].
Definition at line 273 of file libtrac.h.
4.2.2.34 char ctl_t::csi_basename[LEN]
Basename of CSI data files.
Definition at line 276 of file libtrac.h.
4.2.2.35 double ctl_t::csi_dt_out
Time step for CSI data output [s].
Definition at line 279 of file libtrac.h.
4.2.2.36 char ctl_t::csi_obsfile[LEN]
Observation data file for CSI analysis.
Definition at line 282 of file libtrac.h.
4.2.2.37 double ctl_t::csi_obsmin
Minimum observation index to trigger detection.
Definition at line 285 of file libtrac.h.
4.2.2.38 double ctl_t::csi_modmin
Minimum column density to trigger detection [kg/m<sup>2</sup>].
Definition at line 288 of file libtrac.h.
4.2.2.39 int ctl_t::csi_nz
Number of altitudes of gridded CSI data.
Definition at line 291 of file libtrac.h.
```

4.2.2.40 double ctl_t::csi_z0 Lower altitude of gridded CSI data [km]. Definition at line 294 of file libtrac.h. 4.2.2.41 double ctl_t::csi_z1 Upper altitude of gridded CSI data [km]. Definition at line 297 of file libtrac.h. 4.2.2.42 int ctl_t::csi_nx Number of longitudes of gridded CSI data. Definition at line 300 of file libtrac.h. 4.2.2.43 double ctl_t::csi_lon0 Lower longitude of gridded CSI data [deg]. Definition at line 303 of file libtrac.h. 4.2.2.44 double ctl_t::csi_lon1 Upper longitude of gridded CSI data [deg]. Definition at line 306 of file libtrac.h. 4.2.2.45 int ctl_t::csi_ny Number of latitudes of gridded CSI data. Definition at line 309 of file libtrac.h. 4.2.2.46 double ctl_t::csi_lat0 Lower latitude of gridded CSI data [deg]. Definition at line 312 of file libtrac.h. 4.2.2.47 double ctl_t::csi_lat1 Upper latitude of gridded CSI data [deg]. Definition at line 315 of file libtrac.h. 4.2.2.48 char ctl_t::grid_basename[LEN]

Basename of grid data files.

Definition at line 318 of file libtrac.h.

```
4.2.2.49 char ctl_t::grid_gpfile[LEN]
Gnuplot file for gridded data.
Definition at line 321 of file libtrac.h.
4.2.2.50 double ctl_t::grid_dt_out
Time step for gridded data output [s].
Definition at line 324 of file libtrac.h.
4.2.2.51 int ctl_t::grid_sparse
Sparse output in grid data files (0=no, 1=yes).
Definition at line 327 of file libtrac.h.
4.2.2.52 int ctl_t::grid_nz
Number of altitudes of gridded data.
Definition at line 330 of file libtrac.h.
4.2.2.53 double ctl_t::grid_z0
Lower altitude of gridded data [km].
Definition at line 333 of file libtrac.h.
4.2.2.54 double ctl_t::grid_z1
Upper altitude of gridded data [km].
Definition at line 336 of file libtrac.h.
4.2.2.55 int ctl_t::grid_nx
Number of longitudes of gridded data.
Definition at line 339 of file libtrac.h.
4.2.2.56 double ctl_t::grid_lon0
Lower longitude of gridded data [deg].
Definition at line 342 of file libtrac.h.
4.2.2.57 double ctl_t::grid_lon1
Upper longitude of gridded data [deg].
Definition at line 345 of file libtrac.h.
```

```
4.2.2.58 int ctl_t::grid_ny
Number of latitudes of gridded data.
Definition at line 348 of file libtrac.h.
4.2.2.59 double ctl_t::grid_lat0
Lower latitude of gridded data [deg].
Definition at line 351 of file libtrac.h.
4.2.2.60 double ctl_t::grid_lat1
Upper latitude of gridded data [deg].
Definition at line 354 of file libtrac.h.
4.2.2.61 char ctl_t::prof_basename[LEN]
Basename for profile output file.
Definition at line 357 of file libtrac.h.
4.2.2.62 char ctl_t::prof_obsfile[LEN]
Observation data file for profile output.
Definition at line 360 of file libtrac.h.
4.2.2.63 int ctl_t::prof_nz
Number of altitudes of gridded profile data.
Definition at line 363 of file libtrac.h.
4.2.2.64 double ctl_t::prof_z0
Lower altitude of gridded profile data [km].
Definition at line 366 of file libtrac.h.
4.2.2.65 double ctl_t::prof_z1
Upper altitude of gridded profile data [km].
Definition at line 369 of file libtrac.h.
4.2.2.66 int ctl_t::prof_nx
Number of longitudes of gridded profile data.
Definition at line 372 of file libtrac.h.
```

```
4.2.2.67 double ctl_t::prof_lon0
Lower longitude of gridded profile data [deg].
Definition at line 375 of file libtrac.h.
4.2.2.68 double ctl_t::prof_lon1
Upper longitude of gridded profile data [deg].
Definition at line 378 of file libtrac.h.
4.2.2.69 int ctl_t::prof_ny
Number of latitudes of gridded profile data.
Definition at line 381 of file libtrac.h.
4.2.2.70 double ctl_t::prof_lat0
Lower latitude of gridded profile data [deg].
Definition at line 384 of file libtrac.h.
4.2.2.71 double ctl_t::prof_lat1
Upper latitude of gridded profile data [deg].
Definition at line 387 of file libtrac.h.
4.2.2.72 char ctl_t::stat_basename[LEN]
Basename of station data file.
Definition at line 390 of file libtrac.h.
4.2.2.73 double ctl_t::stat_lon
Longitude of station [deg].
Definition at line 393 of file libtrac.h.
4.2.2.74 double ctl_t::stat_lat
Latitude of station [deg].
Definition at line 396 of file libtrac.h.
```

```
4.2.2.75 double ctl_t::stat_r
Search radius around station [km].
Definition at line 399 of file libtrac.h.
The documentation for this struct was generated from the following file:
    · libtrac.h
     met_t Struct Reference
Meteorological data.
#include <libtrac.h>
Data Fields
    • double time
           Time [s].
    • int nx
          Number of longitudes.

    int ny

          Number of latitudes.
    • int np
          Number of pressure levels.
    · double lon [EX]
          Longitude [deg].

    double lat [EY]

          Latitude [deg].

    double p [EP]

          Pressure [hPa].

    double ps [EX][EY]

          Surface pressure [hPa].
    • 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 436 of file libtrac.h.
4.3.2 Field Documentation
4.3.2.1 double met_t::time
Time [s].
Definition at line 439 of file libtrac.h.
4.3.2.2 int met_t::nx
Number of longitudes.
Definition at line 442 of file libtrac.h.
4.3.2.3 int met_t::ny
Number of latitudes.
Definition at line 445 of file libtrac.h.
4.3.2.4 int met_t::np
Number of pressure levels.
Definition at line 448 of file libtrac.h.
4.3.2.5 double met_t::lon[EX]
Longitude [deg].
Definition at line 451 of file libtrac.h.
4.3.2.6 double met_t::lat[EY]
Latitude [deg].
Definition at line 454 of file libtrac.h.
4.3.2.7 double met_t::p[EP]
Pressure [hPa].
Definition at line 457 of file libtrac.h.
```

5 File Documentation 19

```
4.3.2.8 double met_t::ps[EX][EY]
Surface pressure [hPa].
Definition at line 460 of file libtrac.h.
4.3.2.9 float met_t::t[EX][EY][EP]
Temperature [K].
Definition at line 463 of file libtrac.h.
4.3.2.10 float met_t::u[EX][EY][EP]
Zonal wind [m/s].
Definition at line 466 of file libtrac.h.
4.3.2.11 float met_t::v[EX][EY][EP]
Meridional wind [m/s].
Definition at line 469 of file libtrac.h.
4.3.2.12 float met_t::w[EX][EY][EP]
Vertical wind [hPa/s].
Definition at line 472 of file libtrac.h.
4.3.2.13 float met_t::h2o[EX][EY][EP]
Water vapor volume mixing ratio [1].
Definition at line 475 of file libtrac.h.
4.3.2.14 float met_t::o3[EX][EY][EP]
Ozone volume mixing ratio [1].
Definition at line 478 of file libtrac.h.
The documentation for this struct was generated from the following file:
    · libtrac.h
```

5 File Documentation

5.1 center.c File Reference

Calculate center of mass of air parcels.

Functions

• int main (int argc, char *argv[])

5.1.1 Detailed Description

Calculate center of mass of air parcels.

Definition in file center.c.

5.1.2 Function Documentation

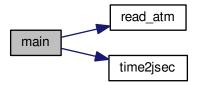
5.1.2.1 int main (int argc, char * argv[])

Definition at line 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
        double latm, lats, lonm, lons, t, zm, zs;
00040
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... */
        fprintf(out,
    "# $1 = time [s]\n"
00059
00060
00061
                 "# $2 = altitude (mean) [km]\n"
00062
                 "# $3 = altitude (sigma) [km] \n"
                 "# $4 = altitude (minimum) [km] \n"
00063
                 "# $5 = altitude (10%% percentile) [km]\n"
00064
00065
                 "# $6 = altitude (1st quarter) [km]\n"
                 "# $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
                 "# $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
                 "# $17 = longitude (3rd quarter) [deg]\n"
"# $18 = longitude (90%% percentile) [deg]\n"
00077
00078
                "# $19 = longitude (maximum) [deg]\n");
00079
08000
       fprintf(out,
00081
                 "# $20 = latitude (mean) [deg] \n
                "# $21 = latitude (sigma) [deg]\n"
00082
                 "# $22 = latitude (minimum) [deg] \n"
00083
                "# $23 = latitude (10%% percentile) [deg]\n"
00084
00085
                "# $24 = latitude (1st quarter) [deg]\n
                "# $25 = latitude (median) [deg]\n"
00086
```

```
"# $26 = latitude (3rd quarter) [deg] n"
00088
                      "# $27 = latitude (90%% percentile) [deg]\n"
                      "# $28 = latitude (maximum) [deg]n";
00089
00090
           /* Loop over files... */
00091
00092
          for (f = 2; f < argc; f++) {</pre>
00094
              /\star Read atmopheric data... \star/
00095
             read_atm(argv[f], &ctl, atm);
00096
00097
             /* Initialize... */
00098
             zm = zs = 0;
             lonm = lons = 0;
00099
00100
             latm = lats = 0;
00101
00102
              /\star Calculate mean and standard deviation... \star/
             for (ip = 0; ip < atm->np; ip++) {
  zm += Z(atm->p[ip]) / atm->np;
00103
00104
                lonm += atm->lon[ip] / atm->np;
00105
                latm += atm->lat[ip] / atm->np;
00106
00107
                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... */
00122
             for (i = (int) strlen(argv[f]) - 1; argv[f][i] != '/' || i == 0; i--);
name = strtok(&(argv[f][i]), "_");
00123
             year = strtok(NULL, "_");
mon = strtok(NULL, "_");
day = strtok(NULL, "_");
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
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 , _,,,, Z(atm->p[atm->np / 4]), Z(atm->p[atm->np / 4]), Z(atm->p[atm->np / 10]), Z(atm->p[0]),
00140
00141
                        lonm, lons, atm->lon[0], atm->lon[atm->np / 10], atm->lon[atm->np / 4], atm->lon[atm->np / 2],
00142
                        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
                        latm, lats, atm->lat[0], atm->lat[atm->np / 10],
00147
                        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
00150
00151
00152
          /* Close file... */
00153
          fclose(out);
00154
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
         MPTRAC is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by
00004
00005
00006
         the Free Software Foundation, either version 3 of the License, or
00007
         (at your option) any later version.
00008
00009
         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 #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
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... */
         if (argc < 3)
00048
00049
           ERRMSG("Give parameters: <outfile> <atm1> [<atm2> ...]");
00050
00051
         /* Write info... */
00052
         printf("Write center of mass data: sn'', argv[1]);
00053
         /* Create output file... */
if (!(out = fopen(argv[1], "w")))
00054
00055
00056
           ERRMSG("Cannot create file!");
00057
00058
         /* Write header... */
         fprintf(out, "# $1
00059
00060
                         = time [s]\n"
                  "# $2 = altitude (mean) [km]\n"
00061
00062
                  "# $3 = altitude (sigma) [km]\n"
                  "# $4 = altitude (minimum) [km] \n"
00063
```

5.2 center.c 23

```
"# $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"
00067
                  "# $9 = altitude (90%% percentile) [km] \n"
00068
00069
                  "# $10 = altitude (maximum) [km] \n");
00070
        fprintf(out,
00071
                  "# $11 = longitude (mean) [deg]\n
00072
                  "# $12 = longitude (sigma) [deg] n"
                  "# $13 = longitude (minimum) [deg]\n"
00073
                  "# $14 = longitude (10%% percentile) [deg]\n"
00074
00075
                  "# $15 = longitude (1st quarter) [deg] n
00076
                  "# $16 = longitude (median) [deg]\n"
00077
                  "# $17 = longitude (3rd quarter) [deg]\n"
00078
                  "# $18 = longitude (90%% percentile) [deg]\n"
00079
                  "# $19 = longitude (maximum) [deg]\n");
        fprintf(out,
    "# $20 = latitude (mean) [deg]\n
08000
00081
                  "# $21 = latitude (sigma) [deg]\n"
00082
00083
                  "# $22 = latitude (minimum) [deg]\n'
                  "# $23 = latitude (10%% percentile) [deg]\n"
00084
                  "# $24 = latitude (1st quarter) [deg] \n"
00085
                  "# $25 = latitude (median) [deg] n"
00086
                  "# $26 = latitude (3rd quarter) [deg] \n"
00087
00088
                  "# $27 = latitude (90%% percentile) [deg]\n"
                  "# $28 = latitude (maximum) [deg] \n\n");
00089
00090
        /* Loop over files... */
for (f = 2; f < argc; f++) {</pre>
00091
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;
00102
            /* Calculate mean and standard deviation... */
00103
           for (ip = 0; ip < atm->np; ip++) {
             zm += Z(atm->p[ip]) / atm->np;
00104
             lonm += atm->lon[ip] / atm->np;
latm += atm->lat[ip] / atm->np;
00105
00106
00107
             zs += gsl_pow_2(Z(atm->p[ip])) / atm->np;
              lons += gsl_pow_2(atm->lon[ip]) / atm->np;
00108
00109
              lats += gsl_pow_2(atm->lat[ip]) / atm->np;
00110
00111
           /* Normalize... */
00112
           zs = sqrt(zs - gsl_pow_2(zm));
00113
           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
00122
           /\star Get date from filename... \star/
           for (i = (int) strlen(argv[f]) - 1; argv[f][i] != '/' || i == 0; i--);
name = strtok(&(argv[f][i]), "_");
00123
00124
           year = strtok(NULL, "_");
mon = strtok(NULL, "_");
day = strtok(NULL, "_");
00125
00126
00127
           hour = strtok(NULL, "_");
00128
           name = strtok(NULL, "_");
min = strtok(name, ".");
00129
                                           /* TODO: Why another "name" here? */
00130
00131
           time2jsec(atoi(year), atoi(mon), atoi(day), atoi(hour), atoi(min), 0, 0,
00132
                      &t);
00133
00134
           /* Write data... */
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]),
00138
00139
                    Z(atm->p[atm->np / 2]), Z(atm->p[atm->np / 4]), Z(atm->p[atm->np / 10]), Z(atm->p[0]),
00140
00141
                    lonm, lons, atm->lon[0], atm->lon[atm->np / 10],
00142
                    atm->lon[atm->np / 4], atm->lon[atm->np / 2],
atm->lon[atm->np - atm->np / 4],
00143
00144
                    atm->lon[atm->np - atm->np / 10],
00145
00146
                    atm->lon[atm->np - 1],
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],
00148
00149
                    atm->lat[atm->np - atm->np / 10], atm->lat[atm->np - 1]);
00150
```

```
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
                           {
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
         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... */
         ALLOC(atm1, atm_t, 1);
ALLOC(atm2, atm_t, 1);
ALLOC(lon1, double,
00047
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);
         ALLOC(lon2, double,
00059
00060
                NP);
00061
         ALLOC(lat2, double,
00062
                NP);
00063
         ALLOC(p2, double,
```

```
00064
               NP);
00065
        ALLOC(1h2, double,
00066
              NP);
        ALLOC(1v2, double,
00067
00068
              NP);
00069
        ALLOC (dh, double,
00070
              NP);
00071
        ALLOC(dv, double,
00072
              NP);
00073
00074
        /* Check arguments... */
00075
        if (argc < 4)
00076
         ERRMSG
00077
             ("Give parameters: <outfile> <atmla> <atmlb> [<atm2a> <atm2b> ...]");
00078
00079
        /* Write info... */
        printf("Write transport deviations: sn", argv[1]);
08000
00081
        /* Create output file... */
if (!(out = fopen(argv[1], "w")))
00082
00083
00084
          ERRMSG("Cannot create file!");
00085
00086
        /* Write header... */
00087
        fprintf(out, "# $1
00088
                       = time [s]\n"
                 "# $2
                       = AHTD (mean) [km]\n"
00089
                       = AHTD (sigma) [km]\n"
00090
                 "# $3
                 "# $4 = AHTD (minimum) [km] \n"
00091
                 "# $5 = AHTD (10%% percentile) [km]\n"
00092
                 "# $6 = AHTD (1st quartile) [km]\n"
00093
00094
                 "# $7 = AHTD (median) [km]\n"
                "# $8 = AHTD (3rd quartile) [km]\n"
"# $9 = AHTD (90%% percentile) [km]\n"
00095
00096
00097
                 "# $10 = AHTD (maximum) [km] \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
00109
                 "# $22 = AVTD (maximum) [km] \n"
                00110
00111
00112
00113
        /* Loop over file pairs... */
00114
        for (f = 2; f < argc; f += 2) {
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
              ERRMSG("Times do not match!");
00125
00126
00127
00128
          ahtd = ahtd2 = 0;
00129
          avtd = avtd2 = 0;
          rhtd = rhtd2 = 0:
00130
00131
          rvtd = rvtd2 = 0;
00132
00133
          /* Loop over air parcels... */
00134
          for (ip = 0; ip < atml->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]));
            avtd += dv[ip];
00146
00147
            avtd2 += gsl_pow_2(dv[ip]);
00148
00149
             /\star Calculate relative transport deviations... \star/
            if (f > 2) {
00150
```

```
00152
                   /* Get trajectory lengths... */
                  geo2cart(0, lon1[ip], lat1[ip], x0);
lh1[ip] += DIST(x0, x1);
00153
00154
                  lv1[ip] += fabs(Z(p1[ip]) - Z(atm1->p[ip]));
00155
00156
                  geo2cart(0, lon2[ip], lat2[ip], x0);
lh2[ip] += DIST(x0, x2);
00157
00158
00159
                  lv2[ip] += fabs(Z(p2[ip]) - Z(atm2->p[ip]));
00160
00161
                   /* Get relative transport devations... */
                  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
                   if (lv1[ip] + lv2[ip] > 0) {
00167
00168
                     aux =
00169
                       200. * fabs(Z(atm1->p[ip]) - Z(atm2->p[ip])) / (lv1[ip] +
00170
00171
00172
                     rvtd2 += gsl_pow_2(aux);
                  }
00173
00174
00175
00176
                /* Save positions of air parcels... */
               lon1[ip] = atml->lon[ip];
lat1[ip] = atml->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
             /\star Get indices of trajectories with maximum errors... \star/
00186
             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
00189
00190
             /* Sort distances to calculate percentiles... */
00191
             gsl_sort(dh, 1, (size_t) atm1->np);
             gsl_sort(dv, 1, (size_t) atml->np);
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, "_");
00200
             name = strtok(NULL, "_");
min = strtok(name, ".");
                                                  /* 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... */
             00207
00208
                        " %g %g %g %g %g %g %g %g %d %g %g\n", t,
00209
                        ahtd / atm1->np,
                        sqrt(ahtd2 / atm1->np - gsl_pow_2(ahtd / atm1->np)),
00210
                       sqrt(antd2 / atmi->np - gsi_pow_2(antd / atmi->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 - 1], iph, rhtd / atml->np,
sqrt(rhtd2 / atml->np - gsl_pow_2(rhtd / atml->np)),
00211
00212
00213
00214
                        avtd / atm1->np,
00215
                       avet / atml >np,
sqrt(avtd2 / atml->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,
sqrt(rvtd2 / atml->np - gsl_pow_2(rvtd / atml->np)));
00216
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);
```

5.4 dist.c 27

```
00238 free(lv2);

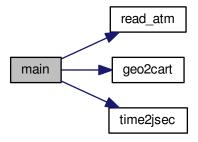
00239 free(dh);

00240 free(dv);

00241 return EXIT_SUCCESS;

00243 }
```

Here is the call graph for this function:



5.4 dist.c

```
00001 /*
00002
         This file is part of MPTRAC.
00003
00004
         \ensuremath{\mathsf{MPTRAC}} is free software: you can redistribute it and/or modify
         it under the terms of the GNU General Public License as published by
00005
         the Free Software Foundation, either version 3 of the License, or
00006
00007
         (at your option) any later version.
80000
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 #include <gsl/gsl_sort.h>
00027
00028 int main(
00029
         int argc,
        char *argv[]) {
00030
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
         double aux, x0[3], x1[3], x2[3], \star lon1, \star lat1, \star p1, \star lh1, \star lv1,
00040
           *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);
ALLOC(atm2, atm_t, 1);
ALLOC(lon1, double,
00048
00049
00050
                NP);
```

```
ALLOC(lat1, double,
00052
              NP);
        ALLOC(p1, double,
00053
00054
               NP);
00055
        ALLOC(lh1, double,
00056
               NP);
        ALLOC(lv1, double,
00058
               NP);
00059
       ALLOC(lon2, double,
00060
              NP);
       ALLOC(lat2, double,
00061
00062
              NP);
00063
       ALLOC(p2, double,
00064
               NP);
00065
        ALLOC(1h2, double,
00066
              NP);
        ALLOC(1v2, double,
00067
00068
               NP);
        ALLOC(dh, double,
00069
00070
               NP);
00071
        ALLOC(dv, double,
00072
              NP);
00073
00074
        /* Check arguments... */
00075
        if (argc < 4)
00076
00077
             ("Give parameters: <outfile> <atmla> <atmlb> [<atm2a> <atm2b> ...]");
00078
00079
        /* Write info... */
        printf("Write transport deviations: %s\n", argv[1]);
00080
00081
00082
        /* Create output file... */
        if (!(out = fopen(argv[1], "w")))
00083
00084
          ERRMSG("Cannot create file!");
00085
        /* Write header... */
00086
00087
        fprintf(out,
                        = time [s]\n"
00089
                 "# $2
                       = AHTD (mean) [km] n"
00090
                 "# $3 = AHTD (sigma) [km] \n"
                 "# $4 = AHTD (minimum) [km] \n"
00091
                 "# $5 = AHTD (10%% percentile) [km]\n"
00092
                 "# $6 = AHTD (1st quartile) [km]\n"
00093
00094
                 "# $7
                       = AHTD (median) [km]\n'
00095
                 "# $8 = AHTD (3rd quartile) [km]\n"
00096
                 "# $9 = AHTD (90%% percentile) [km]\n"
00097
                 "# $10 = AHTD (maximum) [km] \n"
                "# $11 = AHTD (maximum trajectory index)\n"
"# $12 = RHTD (mean) [%*]\n" "# $13 = RHTD (sigma) [%*]\n");
00098
00099
00100
        fprintf(out,
                 "# $14 = AVTD (mean) [km]\n"
00101
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"
00108
                 "# $21 = AVTD (90%% percentile) [km]\n"
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) {
00115
00116
          /* Read atmopheric data... */
00117
          read_atm(argv[f], &ctl, atm1);
read_atm(argv[f + 1], &ctl, atm2);
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
00128
          ahtd = ahtd2 = 0;
          avtd = avtd2 = 0;
00129
          rhtd = rhtd2 = 0;
00130
          rvtd = rvtd2 = 0;
00131
00132
00133
           /* Loop over air parcels... */
00134
          for (ip = 0; ip < atm1->np; ip++) {
00135
            /* Get Cartesian coordinates... */
00136
            geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00137
```

5.4 dist.c 29

```
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
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) {
00151
00152
                    /\star Get trajectory lengths...
                   geo2cart(0, lon1[ip], lat1[ip], x0);
lh1[ip] += DIST(x0, x1);
lv1[ip] += fabs(Z(p1[ip]) - Z(atm1->p[ip]));
00153
00154
00155
00156
                   geo2cart(0, lon2[ip], lat2[ip], x0);
lh2[ip] += DIST(x0, x2);
lv2[ip] += fabs(Z(p2[ip]) - Z(atm2->p[ip]));
00157
00158
00159
00160
00161
                    /\star Get relative transport devations... \star/
                   /* Get lelative transport develops... */
if (lh1[ip] + lh2[ip] > 0) {
  aux = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00162
00163
00164
                      rhtd += aux;
00165
                      rhtd2 += gsl_pow_2(aux);
00166
                   if (lv1[ip] + lv2[ip] > 0) {
00167
00168
                      aux =
00169
                        200. * fabs(Z(atm1->p[ip]) - Z(atm2->p[ip])) / (lv1[ip] +
00170
00171
                      rvtd += aux;
00172
                      rvtd2 += gsl_pow_2(aux);
00173
00174
                }
00175
00176
                 /* Save positions of air parcels... */
                lon1[ip] = atml->lon[ip];
lat1[ip] = atml->lat[ip];
00177
00178
00179
                p1[ip] = atm1->p[ip];
00180
00181
                 lon2[ip] = atm2->lon[ip];
                 lat2[ip] = atm2->lat[ip];
00182
00183
                p2[ip] = atm2->p[ip];
00184
00185
00186
              /* Get indices of trajectories with maximum errors... */
             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
00190
              /\star Sort distances to calculate percentiles... \star/
             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]), "_");
year = strtok(NULL, "_");
mon = strtok(NULL, "_");
day = strtok(NULL, "_");
00195
00196
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
              /* Write output... */
00206
             00207
00208
00209
                         ahtd / atm1->np,
                         antd / atml >np,
sqrt(ahtd2 / atml ->np - gsl_pow_2(ahtd / atml ->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 - 1], iph, rhtd / atml ->np,
00210
00211
00212
00213
00214
                         sqrt(rhtd2 / atm1->np - gsl_pow_2(rhtd / atm1->np)),
00215
                         avtd / atm1->np,
                         avetd / atml=>np,
sqrt(avtd2 / atml=>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);
```

```
00226
        /* Free... */
00227
       free(atm1);
00228
       free(atm2);
00229
       free (lon1);
00230
        free(lat1);
00231
       free(p1);
00232
00233
       free(lv1);
00234
        free(lon2);
00235
        free (lat2);
00236
       free (p2);
00237
        free (lh2);
00238
00239
        free(dh);
00240
       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[])

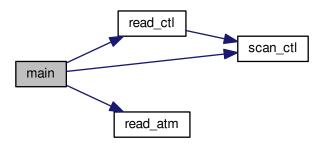
Definition at line 28 of file extract.c.

```
00030
00031
00032
        ctl_t ctl;
00033
00034
        atm t *atm;
00035
00036
        FILE *in, *out;
00037
00038
00039
        int f, ip, iq;
00040
        /* Allocate... */
00041
        ALLOC(atm, atm_t, 1);
00042
00043
        /* Check arguments... */
00044
          ERRMSG("Give parameters: <ctl> <outfile> <atml> [<atm2> ...]");
00045
00046
00047
        /* Read control parameters... */
00048
        read_ctl(argv[1], argc, argv, &ctl);
00049
        ip = (int) scan_ctl(argv[1], argc, argv, "EXTRACT_IP", -1, "0", NULL);
00050
00051
       /* Write info... */
printf("Write trajectory data: %s\n", argv[2]);
00052
00053
00054
        /* Create output file... */
```

5.6 extract.c 31

```
if (!(out = fopen(argv[2], "w")))
00056
          ERRMSG("Cannot create file!");
00057
00058
        /* Write header... */
00059
        00060
                "# $2 = altitude [km] \n"
00061
                "# $3 = longitude [deg] \n" "# $4 = latitude [deg] \n");
00062
       00063
00064
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
00078
          /* Write data... */
fprintf(out, "%.2f %g %g %g", atm->time[ip],
00079
00080
                 Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
          for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00081
00082
00083
00084
00085
          fprintf(out, "\n");
00086
00087
88000
        /* Close file... */
00089
       fclose(out);
00090
00091
        /* Free... */
00092
       free(atm);
00093
00094
        return EXIT_SUCCESS;
00095 }
```

Here is the call graph for this function:



5.6 extract.c

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

```
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
        GNU General Public License for more details.
00012
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,
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)
          ERRMSG("Give parameters: <ctl> <outfile> <atml> [<atm2> ...]");
00045
00046
00047
        /* Read control parameters... */
00048
        read_ctl(argv[1], argc, argv, &ctl);
00049
        ip = (int) scan_ctl(argv[1], argc, argv, "EXTRACT_IP", -1, "0", NULL);
00050
00051
        /* Write info... */
00052
        printf("Write trajectory data: %s\n", argv[2]);
00053
00054
        /* Create output file...
00055
        if (!(out = fopen(argv[2], "w")))
          ERRMSG("Cannot create file!");
00056
00057
00058
        /* Write header... */
00059
        fprintf(out,
00060
                 "# $1 = time [s] \n"
                 "# $2 = altitude [km] \n"
00061
00062
                "# $3 = longitude [deg] \n" "# <math>$4 = latitude [deg] \n");
        00063
00064
00065
        fprintf(out, "\n");
00066
00067
00068
        /* Loop over files... ∗/
00069
        for (f = 3; f < argc; f++) {</pre>
00070
00071
          /* Read atmopheric data... */
if (!(in = fopen(argv[f], "r")))
00072
00073
            continue;
00074
          else
00075
            fclose(in);
          read_atm(argv[f], &ctl, atm);
00076
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
        /* Close file... */
00088
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.

5.7 init.c File Reference 33

Functions

• int main (int argc, char *argv[])

5.7.1 Detailed Description

Create atmospheric data file with initial air parcel positions.

Definition in file init.c.

5.7.2 Function Documentation

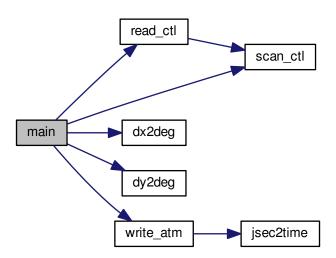
```
5.7.2.1 int main (int argc, char * argv[])
```

Definition at line 27 of file init.c.

```
00029
00030
00031
                  atm_t *atm;
00032
00033
                 ctl t ctl;
00034
                 gsl_rng *rng;
00036
00037
                 double dt, dz, dlon, dlat, lat0, lat1, lon0, lon1, t0, t1, z0, z1,
00038
                    t, z, lon, lat, st, sz, slon, slat, sx, ut, uz, ulon, ulat, m;
00039
00040
                 int ip, irep, rep;
00041
00042
                    /* Allocate... */
00043
                 ALLOC(atm, atm_t, 1);
00044
00045
                  /* Check arguments... */
00046
                  if (argc < 3)
00047
                      ERRMSG("Give parameters: <ctl> <atm_out>");
00048
00049
                 /* Read control parameters... ∗/
                 read_ctl(argv[1], argc, argv, &ctl);
t0 = scan_ctl(argv[1], argc, argv, "INIT_TO", -1, "0", NULL);
t1 = scan_ctl(argv[1], argc, argv, "INIT_T1", -1, "0", NULL);
dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
00050
00051
00052
00053
                 dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
z0 = scan_ctl(argv[1], argc, argv, "INIT_20", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "INIT_Z1", -1, "0", NULL);
dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);
lon0 = scan_ctl(argv[1], argc, argv, "INIT_LON0", -1, "0", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "INIT_LON1", -1, "0", NULL);
dlon = scan_ctl(argv[1], argc, argv, "INIT_LAT0", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT0", -1, "0", NULL);
00054
00055
00056
00057
00058
00059
                lat0 = scan_ctl(argv[1], argc, argv, "INIT_LATO", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT1", -1, "0", NULL);
dlat = scan_ctl(argv[1], argc, argv, "INIT_DLAT1", -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_SLON", -1, "0", NULL);
slat = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
sx = scan_ctl(argv[1], argc, argv, "INIT_SX", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_UT", -1, "0", NULL);
ulon = scan_ctl(argv[1], argc, argv, "INIT_UZ", -1, "0", NULL);
ulon = scan_ctl(argv[1], argc, argv, "INIT_ULON", -1, "0", NULL);
ulat = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
rep = (int) scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
00060
00061
00062
00063
00064
00065
00067
00068
00069
00070
00071
00072
00073
00074
00075
                 /* Initialize random number generator... */
00076
                 gsl_rng_env_setup();
00077
                 rng = gsl_rng_alloc(gsl_rng_default);
00078
00079
                   /* Create grid... *
08000
                  for (t = t0; t \le t1; t += dt)
                   for (z = z0; z <= z1; z += dz)
  for (lon = lon0; lon <= lon1; lon += dlon)
  for (lat = lat0; lat <= lat1; lat += dlat)</pre>
00081
00082
00083
00084
                                     for (irep = 0; irep < rep; irep++) {</pre>
```

```
/* 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));
                      atm->p[atm->np]
00090
00091
                         = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
00092
                              + uz * (gsl_rng_uniform(rng) - 0.5));
00093
                      atm->lon[atm->np]
00094
                        = (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
                      atm->lat[atm->np]
00097
                         + default + 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
                      /* Set particle counter... */
if ((++atm->np) >= NP)
   ERRMSG("Too many particles!");
00102
00103
00104
00105
                    }
00106
00107
          /\star Check number of air parcels... \star/
         if (atm->np <= 0)
    ERRMSG("Did not create any air parcels!");</pre>
00108
00109
00110
00111
          /* Initialize mass... */
00112
          if (ctl.qnt_m >= 0)
00113
          for (ip = 0; ip < atm->np; ip++)
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.
```

5.8 init.c 35

```
00004
             MPTRAC is free software: you can redistribute it and/or modify
00005
             it under the terms of the GNU General Public License as published by
             the Free Software Foundation, either version 3 of the License, or
00006
00007
             (at your option) any later version.
00008
             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;
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
00042
              /* Allocate... */
00043
             ALLOC(atm, atm_t, 1);
00044
             /* Check arguments... */
00045
00046
             if (argc < 3)</pre>
00047
                ERRMSG("Give parameters: <ctl> <atm_out>");
00048
             /* Read control parameters...

read_ctl(argv[1], argc, argv, &ctl);

read_ctl(argv[1], argc, argv, "INIT_TO", -1, "0", NULL);
00049
00050
             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);
00051
00052
            dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
z0 = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "INIT_Z1", -1, "0", NULL);
dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);
lon0 = scan_ctl(argv[1], argc, argv, "INIT_LONO", -1, "0", NULL);
dlon1 = scan_ctl(argv[1], argc, argv, "INIT_LONO1", -1, "0", NULL);
dlon2 = scan_ctl(argv[1], argc, argv, "INIT_LONO1", -1, "1", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "INIT_LAT0", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT1", -1, "0", NULL);
dlat = scan_ctl(argv[1], argc, argv, "INIT_DLAT1", -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_SLON", -1, "0", NULL);
00053
00054
00055
00056
00057
00059
00060
00061
00062
00063
            sz = scan_ctl(argv[1], argc, argv, "INIT_SZ", -1, "0", NULL);
slon = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
slat = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
sx = scan_ctl(argv[1], argc, argv, "INIT_SX", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_UT", -1, "0", NULL);
uz = scan_ctl(argv[1], argc, argv, "INIT_UZ", -1, "0", NULL);
ulon = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
ulat = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
rep = (int) scan_ctl(argv[1], argc, argv, "INIT_REP", -1, "1", NULL);
m = scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
00065
00066
00067
00068
00069
00070
00071
00072
00073
00074
00075
             /* Initialize random number generator... */
00076
            asl rna env setup();
00077
             rng = gsl_rng_alloc(gsl_rng_default);
00078
00079
             /* Create grid... */
00080
             for (t = t0; t <= t1; t += dt)</pre>
00081
                for (z = z0; z \le z1; z += dz)
00082
                    for (lon = lon0; lon <= lon1; lon += dlon)</pre>
                        for (lat = lat0; lat <= lat1; lat += dlat)</pre>
00083
00084
                           for (irep = 0; irep < rep; irep++) {</pre>
00085
00086
                               /* Set position... */
00087
                              atm->time[atm->npl
                                 = (t + gsl_ran_gaussian_ziggurat(rng, st / 2.3548)
00088
00089
                                        + ut * (gsl_rng_uniform(rng) - 0.5));
                               atm->p[atm->np]
00090
00091
                                 = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
00092
                                         + uz * (gsl_rng_uniform(rng) - 0.5));
00093
                               atm->lon[atm->np]
00094
                                   = (lon + gsl_ran_gaussian_ziggurat(rng, slon / 2.3548)
```

```
+ gsl_ran_gaussian_ziggurat(rng, dx2deg(sx, lat) / 2.3548)
00096
                           + ulon * (gsl_rng_uniform(rng) - 0.5));
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... */
00103
                      if ((++atm->np) >= NP)
                        ERRMSG("Too many particles!");
00104
                   }
00105
00106
         /* Check number of air parcels... */
00107
00108
         if (atm->np <= 0)
00109
           ERRMSG("Did not create any air parcels!");
00110
         /* Initialize mass... */
00111
         if (ctl.qnt_m >= 0)
  for (ip = 0; ip < atm->np; ip++)
00112
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
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
        if (argc < 2)
         ERRMSG("Give parameters: <jsec>");
00037
00038
00039
       /* Read arguments... */
00040
       jsec = atof(argv[1]);
00041
00042
        /* Convert time... */
00043
        jsec2time(jsec, &year, &mon, &day, &hour, &min, &sec, &remain);
00044
        printf("%d %d %d %d %d %g\n", year, mon, day, hour, min, sec, remain);
00045
00046
        return EXIT_SUCCESS;
00047 }
```

5.10 jsec2time.c 37

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 (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, int np, float dest[EX][EY][EP], float scl)

Read and convert variable from meteorological data file.

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_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.

5.11.2.3 double deg2dy (double dlat)

Convert degrees to horizontal distance.

Definition at line 54 of file libtrac.c.

```
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.

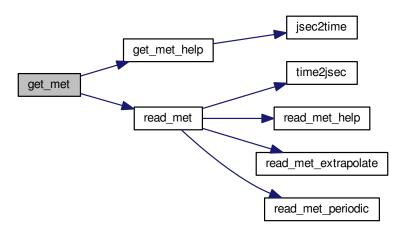
5.11.2.9 void get_met (ctl_t * ctl, char * metbase, double t, met_t * met0, met_t * met1)

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
00133
          read_met(filename, met0);
00134
00135
          get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
dt_met, filename);
00136     read_met(filen
          read_met(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));
00142
         get_met_help(t, 1, metbase, ctl->dt_met, filename);
00143
          read_met(filename, met1);
00144
00145
00146
        /\star Read new data for backward trajectories... \star/
00147
       if (t < met0->time && ctl->direction == -1) {
00148
         memcpy(met1, met0, sizeof(met_t));
00149
          get_met_help(t, -1, metbase, ctl->dt_met, filename);
00150
          read_met(filename, met0);
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.

```
{
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)
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_%02d.nc", metbase, year, mon, day, hour);
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.

```
00188
00189
00190
           double aux00, aux01, aux10, aux11;
00191
00192
           /* Set variables...
          /* Set Variables...*/
aux00 = array[ix][iy];
aux01 = array[ix][iy + 1];
aux10 = array[ix + 1][iy];
aux11 = array[ix + 1][iy + 1];
00193
00194
00195
00196
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
00215
00216
         double aux00, aux01, aux10, aux11;
00217
        /* Interpolate vertically... */
00218
        aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00219
          + array[ix][iy][ip + 1];
00220
00221
        aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
        + array[ix][iy + 1][ip + 1];
aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
00222
00223
        + array[ix + 1][iy][ip + 1];
aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
00224
00225
         + array[ix + 1][iy + 1][ip + 1];
00226
00227
00228
        /* Interpolate horizontally... */
00229    aux00 = wy * (aux00 - aux01) + aux01;
00230    aux11 = wy * (aux10 - aux11) + aux11;
00231
        *var = wx * (aux00 - aux11) + aux11;
00232 }
```

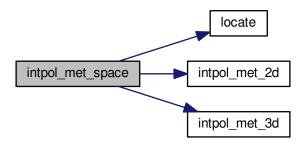
5.11.2.13 void intpol_met_space (met_t * met, double p, double lon, double lon)

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
00253
        /* Check longitude... */
00254
        if (lon < 0)
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... */
00268
        if (ps != NULL)
           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);
         if (w != NULL)
00276
           intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
00277
00278
        if (h2o != NULL)
        intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
if (o3 != NULL)
00279
00280
00281
          intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00282 }
```

Here is the call graph for this function:



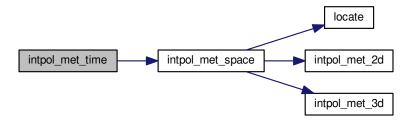
5.11.2.14 void intpol_met_time (met_t * met0, met_t * met1, double ts, double p, double lon, double lat, double * ps, double * t, double * u, double * v, double * w, double * b2o, double * o3)

Temporal interpolation of meteorological data.

Definition at line 286 of file libtrac.c.

```
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
       if (w != NULL)
00331
00332
          \star w = wt \star (w0 - w1) + w1;
       if (h2o != NULL)
00333
00334
         *h2o = wt * (h2o0 - h2o1) + h2o1;
       if (o3 != NULL)
00335
00336
         *o3 = wt * (o30 - o31) + o31;
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
       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;
       t0.tm\_sec = 0;
00360
00361
        jsec0 = (time_t) jsec + timegm(&t0);
00362
00363
       t1 = gmtime(&jsec0);
00364
00365
        *year = t1->tm_year + 1900;
       *mon = t1->tm_mon + 1;
*day = t1->tm_mday;
00366
00367
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;
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
             else
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;
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
          char line[LEN], *tok;
00416
00417
          int iq;
00418
          /* Init... */
atm->np = 0;
00419
00420
00421
00422
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... */
          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
              /\star Convert altitude to pressure... \star/
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 }
```

5.11.2.18 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
        int iq;
00464
00465
00466
        /* Write info... */
         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... */
00472
        ctl->qnt_m = -1;
         ct1\rightarrow qnt_r = -1;
00473
00474
         ctl->qnt_rho =
                           -1;
00475
         ctl->qnt_ps = -1;
00476
         ctl->qnt_t = -1;
         ct1->qnt_u = -1;
00477
         ctl->qnt_v = -1;
00478
         ctl->qnt_w = -1;
00479
00480
         ct1->qnt_h2o = -1;
00481
         ctl->qnt_o3 = -1;
00482
         ctl->qnt\_theta = -1;
00483
         ctl->qnt\_stat = -1;
00484
00485
         /* Read quantities... */
00486
         ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
         for (iq = 0; iq < ctl->nq; iq++) {
00488
           /* 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",
00489
00490
00491
00492
                     ctl->gnt format[ig]);
00493
00494
           /* Try to identify quantity... */
00495
           if (strcmp(ctl->qnt_name[iq], "m") == 0) {
             ctl->qnt_m = iq;
sprintf(ctl->qnt_unit[iq], "kg");
00496
00497
           } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
ctl->qnt_r = iq;
00498
00500
             sprintf(ctl->qnt_unit[iq], "m");
00501
           } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
             ctl->qnt_rho = iq;
00502
00503
             sprintf(ctl->qnt_unit[iq], "kg/m^3");
           } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
00504
             ctl->qnt_ps = iq;
00505
             sprintf(ctl->qnt_unit[iq], "hPa");
00506
00507
           } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
            ctl->qnt_t = iq;
00508
00509
             sprintf(ctl->qnt_unit[iq], "K");
           } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
00510
            ctl->qnt_u = iq;
00511
             sprintf(ctl->qnt_unit[iq], "m/s");
00512
00513
           } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
             ctl->qnt_v = iq;
00514
00515
             sprintf(ctl->qnt_unit[iq], "m/s");
           } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
00516
             ctle=if (stremp(ctr >qnt_name[rq],
ctl->qnt_w = iq;
sprintf(ctl->qnt_unit[iq], "hPa/s");
00517
00519
           } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
             ctl->qnt_h2o = iq;
00520
00521
             sprintf(ctl->qnt_unit[iq], "1");
           } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
  ctl->qnt_o3 = iq;
00522
00523
00524
             sprintf(ctl->qnt_unit[iq], "1");
00525
           } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
00526
             ctl->qnt_theta = iq;
00527
             sprintf(ctl->qnt_unit[iq], "K");
           } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
  ctl->qnt_stat = iq;
00528
00529
00530
             sprintf(ctl->qnt_unit[iq], "-");
          } else
00532
             scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00533
00534
00535
         /* Time steps of simulation... */
00536
         ctl->direction =
00537
           (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
00538
            (ctl->direction != -1 && ctl->direction != 1)
00539
          ERRMSG("Set DIRECTION to -1 or 1!");
00540
         ctl->t_start
        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);
00541
00542
00543
00544
00545
         /* Meteorological data...
00546
        ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
00547
00548
        /* Isosurface parameters... */
```

```
00549
        ctl->isosurf
         = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00550
00551
00552
00553
         /* Diffusion parameters... */
00554
         ctl->turb dx trop
00555
            = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00556
         ctl->turb_dx_strat
00557
           = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00558
         ctl->turb dz trop
           = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00559
00560
         ctl->turb dz strat
00561
            = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00562
         ctl->turb_meso =
00563
           scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00564
00565
        /* Life time of particles... */
        ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
00566
         ctl->tdec_strat =
00567
00568
           scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00569
        /* Output of atmospheric data... */
scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
00570
00571
       atm basename):
00572
         scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00573
         ctl->atm_dt_out =
00574
           scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00575
00576
        /* Output of CSI data... */
        scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
00577
       csi basename):
00578
        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",
00579
00580
00581
                  ctl->csi_obsfile);
         ctl->csi obsmin =
00582
           scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00583
         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);
00585
00586
00587
00588
         ctl->csi_lon0 =
00589
00590
           scan_ctl(filename, argc, argv, "CSI_LONO", -1, "-180", NULL);
         ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
00591
00592
         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);
00593
00594
00595
00596
         ctl->csi nv =
00597
           (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
00598
00599
         /* Output of grid data... */
        00600
00601
         scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
00602
      grid_gpfile);
00603
        ctl->grid_dt_out =
00604
           scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
         ctl->grid_sparse
00605
         (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);
00606
00607
00608
00609
         ctl->grid nz =
00610
           (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00611
         ctl->grid_lon0 =
           scan_ctl(filename, argc, argv, "GRID_LONO", -1, "-180", NULL);
00612
00613
         ctl->grid_lon1 :
00614
           scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
00615
         ctl->grid_nx =
00616
            (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00617
         ctl->grid lat0 =
00618
           scan_ctl(filename, argc, argv, "GRID_LATO", -1, "-90", NULL);
00619
         ctl->grid lat1
           scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
00620
         ctl->grid_ny =
00621
           (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
00622
00623
00624
         /* Output of profile data... */
         scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
00625
00626
                   ctl->prof basename);
00627
         scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
       prof obsfile);
00628
         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);
00629
00630
         ctl->prof nz =
00631
           (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
```

```
00632
        ctl->prof_lon0 =
00633
           scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
00634
         ctl->prof_lon1 =
           scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
00635
00636
         ctl->prof nx =
           (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00637
00638
         ctl->prof_lat0 =
00639
            scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
00640
         ctl->prof_lat1
           scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
00641
00642
         ctl->prof_ny =
           (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00643
00644
00645
         /* Output of station data... */
        scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
00646
                   ctl->stat_basename);
00647
        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);
00648
00649
00650
00651 }
```

Here is the call graph for this function:



5.11.2.19 void read_met (char * filename, met_t * met)

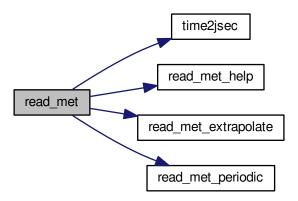
Read meteorological data file.

Definition at line 655 of file libtrac.c.

```
00657
00658
00659
        char tstr[10];
00660
00661
       static float help[EX * EY];
00662
00663
        int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00664
00665
        size_t np, nx, ny;
00666
00667
        /* Write info... */
        printf("Read meteorological data: %s\n", filename);
00668
00669
        /\star Get time from filename... \star/
00671
        sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00672
        year = atoi(tstr);
00673
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00674
        mon = atoi(tstr);
sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00675
00676
        day = atoi(tstr);
00677
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00678
        hour = atoi(tstr);
00679
        time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00680
        /* Open netCDF file... */
00681
00682
        NC(nc_open(filename, NC_NOWRITE, &ncid));
00683
       /* Get dimensions... */
NC(nc_inq_dimid(ncid, "lon", &dimid));
00684
00685
00686
        NC(nc_inq_dimlen(ncid, dimid, &nx));
00687
        if (nx > EX)
00688
          ERRMSG("Too many longitudes!");
00689
```

```
NC(nc_inq_dimid(ncid, "lat", &dimid));
NC(nc_inq_dimlen(ncid, dimid, &ny));
00691
00692
          if (ny > EY)
            ERRMSG("Too many latitudes!");
00693
00694
          NC(nc_inq_dimid(ncid, "lev", &dimid));
NC(nc_inq_dimlen(ncid, dimid, &np));
00695
00697
              (np > EP)
00698
            ERRMSG("Too many pressure levels!");
00699
00700
          /* Store dimensions... */
          met->np = (int) np;
met->nx = (int) nx;
00701
00702
00703
          met->ny = (int) ny;
00704
          /* Read geolocations... */
NC(nc_inq_varid(ncid, "lev", &varid));
00705
00706
00707
          NC(nc_get_var_double(ncid, varid, met->p));
00709
          NC(nc_inq_varid(ncid, "lon", &varid));
00710
          NC(nc_get_var_double(ncid, varid, met->lon));
00711
00712
          NC(nc_inq_varid(ncid, "lat", &varid));
00713
          NC(nc_get_var_double(ncid, varid, met->lat));
00714
00715
           /* Check and convert pressure levels... */
          for (ip = 0; ip < met->np; ip++) {
  if (ip > 0 && met->p[ip - 1] > met->p[ip])
00716
00717
00718
               ERRMSG("Pressure levels must be descending!");
00719
            met->p[ip] /= 100.;
00720
00721
00722
           /* Read surface pressure... */
00723
          if (nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00724
            NC(nc_get_var_float(ncid, varid, help));
             for (iy = 0; iy < met->ny; iy++)
for (ix = 0; ix < met->nx; ix++)
00725
00726
                 met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00728
          } else {
00729
            for (ix = 0; ix < met->nx; ix++)
               for (iy = 0; iy < met->ny; iy++)
  met->ps[ix][iy] = met->p[0];
00730
00731
00732
00733
          /* Read meteorological data... */
read_met_help(ncid, "t", "T", met, met->np, met->t, 1.0);
read_met_help(ncid, "u", "U", met, met->np, met->u, 1.0);
read_met_help(ncid, "v", "V", met, met->np, met->v, 1.0);
read_met_help(ncid, "w", "W", met, met->np, met->w, 0.01f);
read_met_help(ncid, "q", "Q", met, met->np, met->h2o, 1.608f);
00734
00735
00736
00737
00738
00739
00740
          read_met_help(ncid, "o3", "O3", met, met->np, met->o3, 0.602f);
00741
00742
          /\star Extrapolate data for lower boundary... \star/
00743
          read_met_extrapolate(met);
00744
00745
          /\star Copy data to obtain periodic boundary conditions... \star/
00746
          read_met_periodic(met);
00747
00748
           /* Close file... */
00749
          NC(nc_close(ncid));
00750 }
```

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 754 of file libtrac.c.

```
00755
                        {
00756
00757
        int ip, ip0, ix, iy;
00758
00759
        /* Loop over columns... */
00760
        for (ix = 0; ix < met->nx; ix++)
00761
           for (iy = 0; iy < met->ny; iy++) {
00762
00763
             /* Find lowest valid data point... */
             for (ip0 = met->np - 1; ip0 >= 0; ip0--)
   if (!gsl_finite(met->t[ix][iy][ip0])
00764
00765
00766
                    || !gsl_finite(met->u[ix][iy][ip0])
00767
                    || !gsl_finite(met->v[ix][iy][ip0])
00768
                    || !gsl_finite(met->w[ix][iy][ip0]))
00769
                 break:
00770
             /* Extrapolate... */
for (ip = ip0; ip >= 0; ip--) {
00771
00772
00773
               met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
00774
               met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
00775
               met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
               met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
00776
00777
00778
               met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00779
00780
           }
00781 }
```

5.11.2.21 void read_met_help (int *ncid*, char * *varname*, char * *varname*2, met_t * *met*, int *np*, float *dest[EX][EY][EP]*, float *scl*)

Read and convert variable from meteorological data file.

Definition at line 785 of file libtrac.c.

```
00792
                      {
00793
00794
         static float help[EX * EY * EP];
00795
         int ip, ix, iy, n = 0, varid;
00796
00797
00798
         /\star Check if variable exists... \star/
00799
         if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00800
          if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00801
00802
00803
         /* Read data... */
         NC(nc_get_var_float(ncid, varid, help));
00804
00805
00806
         /* Copy and check data...
00807
         for (ip = 0; ip < np; ip++)</pre>
           for (iy = 0; iy < met->ny; iy++)
for (ix = 0; ix < met->nx; ix++) {
00808
00809
               dest[ix][iy][ip] = scl * help[n++];
if (dest[ix][iy][ip] < -le10 || dest[ix][iy][ip] > le10)
00810
00811
00812
                  dest[ix][iy][ip] = GSL_NAN;
00813
00814 }
```

5.11.2.22 void read_met_periodic (met_t * met)

Create meteorological data with periodic boundary conditions.

Definition at line 818 of file libtrac.c.

```
00819
                                {
00820
00821
           int ip, iy;
00822
00823
            /* Check longitudes... */
           if (fabs(met->lon[met->nx - 1] - met->lon[0] - 360) < 0.01)
00824
00825
              return;
00826
00827
           /* Increase longitude counter... */
00828
           if ((++met->nx) > EX)
00829
              ERRMSG("Cannot create periodic boundary conditions!");
00830
          /* Set longitude... */
met->lon[met->nx - 2] + met->lon[1] - met->
00831
00832
        lon[0];
00833
00834
            /* 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];
00835
00836
00837
                 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->v[0][iy][ip];
met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
00838
00839
00840
00841
                 met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
00842
00843
00845 }
```

5.11.2.23 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 849 of file libtrac.c.

```
int contain = 0, i;
00865
          /* Open file... */
         if (filename[strlen(filename) - 1] != '-')
  if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");
00866
00867
00868
00870
         /* Set full variable name... */
00871
         if (arridx >= 0) {
          sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
00872
00873
00874
         } else {
         sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
00875
00876
00877
00878
00879
         /* Read data... */
00880
         if (in != NULL)
           while (fgets(line, LEN, in))
00881
             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
if (strcasecmp(rvarname, fullname1) == 0 ||
00882
00883
00884
                      strcasecmp(rvarname, fullname2) == 0) {
00885
                   contain = 1;
00886
                   break;
00887
                 }
         for (i = 1; i < argc - 1; i++)</pre>
          if (strcasecmp(argv[i], fullname1) == 0 ||
00889
              strcasecmp(argv[i], fullname2) == 0) {
sprintf(rval, "%s", argv[i + 1]);
00890
00891
00892
              contain = 1;
00893
              break:
00894
           }
00895
00896
         /* Close file... */
00897
         if (in != NULL)
           fclose(in);
00898
00899
         /* Check for missing variables... */
00901
         if (!contain) {
         if (strlen(defvalue) > 0)
    sprintf(rval, "%s", defvalue);
00902
00903
           else {
00904
             sprintf(msg, "Missing variable %s!\n", fullname1);
00905
00906
              ERRMSG (msg);
00907
00908
00909
        /* Write info... */
printf("%s = %s\n", fullname1, rval);
00910
00911
00912
         /* Return values... */
00914
         if (value != NULL)
00915
           sprintf(value, "%s", rval);
00916
         return atof(rval);
00917 }
```

5.11.2.24 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 921 of file libtrac.c.

```
00929
00931
       struct tm t0, t1;
00932
00933
       t0.tm_year = 100;
        t0.tm_mon = 0;
00934
       t0.tm_mday = 1;
t0.tm_hour = 0;
00935
00936
00937
        t0.tm_min = 0;
00938
       t0.tm\_sec = 0;
00939
00940
       t1.tm year = year - 1900;
        t1.tm_mon = mon - 1;
00941
00942
        t1.tm_mday = day;
00943
        t1.tm_hour = hour;
        t1.tm_min = min;
00944
       t1.tm_sec = sec;
00945
00946
00947
       *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
00948 }
```

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

Measure wall-clock time.

Definition at line 952 of file libtrac.c.

```
00955
00956
00957
       static double starttime[NTIMER], runtime[NTIMER];
00958
        /* Check id...
00959
00960
        if (id < 0 || id >= NTIMER)
         ERRMSG("Too many timers!");
00961
00962
00963
        /* Start timer... */
00964
       if (mode == 1) {
        if (starttime[id] <= 0)</pre>
00965
00966
            starttime[id] = omp_get_wtime();
00967
           ERRMSG("Timer already started!");
00968
00969
00970
00971
        /* Stop timer... */
00972
        else if (mode == 2) {
00973
         if (starttime[id] > 0) {
00974
           runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
            starttime[id] = -1;
00975
00976
         } else
00977
            ERRMSG("Timer not started!");
00978
00979
        /* Print timer... ∗/
00980
       else if (mode == 3)
00981
         printf("%s = %g s\n", name, runtime[id]);
00982
```

5.11.2.26 double tropopause (double t, double lat)

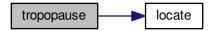
Definition at line 987 of file libtrac.c.

```
00989
00990
00991
            static double doys[12]
             = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00992
00993
00994
             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,
00995
00996
               -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,
00997
00998
01000
01001
                75, 77.5, 80, 82.5, 85, 87.5, 90
01002
            };
01003
            static double tps[12][73]
01004
             = { {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,
01005
                         175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
01007
01008
                         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,
01009
01010
01011
             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,
01012
01013
01014
              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, 98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27, 98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193, 220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
01015
01016
01017
01019
              284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01020
               287.5, 286.2, 285.8},
01021
             {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, 100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01022
01023
01024
              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,
01027
01028
            304.3, 304.9, 306, 306.6, 306.2, 306},
01029
          {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,
01030
01031
           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,
01033
01034
           148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
01035
           263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4
          315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1, 
{266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8, 
260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
01036
01037
                    258.6,
           205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
01039
01040
           101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
           102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9, 165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
01041
01042
           273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01043
           325.3, 325.8, 325.8},
          {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
01045
           222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
01046
01047
           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, 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,
01048
01049
01050
           308.5, 312.2, 313.1, 313.3},
01052
01053
          {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
           187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6, 235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1, 110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7, 111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
01054
01055
01056
           117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
01058
01059
           224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5
01060
           275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
          {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01061
           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,
01062
01064
            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
01065
01066
           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,
01067
          278.2, 282.6, 287.4, 290.9, 292.5, 293}, {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
01068
01069
           183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
01070
01071
           243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
01072
           114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
           110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6, 114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9, 203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
01073
01074
01075
            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,
01077
01078
           215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
           237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2, 111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01079
01080
           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,
01081
           206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01083
           279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
01084
           305.1},
01085
          {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01086
           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,
01087
           108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5,
           102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
01090
           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,
01091
01092
           286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6}
01093
          {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
01094
           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,
01096
01097
           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,
01098
01099
01100
           280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
           281.7, 281.1, 281.2}
01101
01102
01103
01104
          double doy, p0, p1, pt;
01105
01106
          int imon, ilat;
          /* Get day of year... */
doy = fmod(t / 86400., 365.25);
01108
01109
01110
          while (doy < 0)
01111
            dov += 365.25;
01112
```

```
01113
         /* Get indices... */
         imon = locate(doys, 12, doy);
ilat = locate(lats, 73, lat);
01114
01115
01116
         /* Get tropopause pressure... */
01117
         p0 = LIN(lats[ilat], tps[imon][ilat],
lats[ilat + 1], tps[imon][ilat + 1], lat);
01118
01119
         p1 = LIN(lats[ilat], tps[imon + 1][ilat],
lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
01120
01121
01122
         pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01123
01124
         /* Return tropopause pressure... */
01125
         return pt;
01126 }
```

Here is the call graph for this function:



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

Write atmospheric data.

Definition at line 1130 of file libtrac.c.

```
01134
01135
01136
        FILE *in, *out;
01137
01138
        char line[LEN];
01139
01140
        double r;
01141
01142
         int ip, iq, year, mon, day, hour, min, sec;
01143
         /* Check if gnuplot output is requested... */ if (ctl->atm_gpfile[0] != '-') {
01144
01145
01146
01147
           /\star Write info... \star/
01148
           printf("Plot atmospheric data: %s.png\n", filename);
01149
          /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
01150
01151
             ERRMSG("Cannot create pipe to gnuplot!");
01152
01153
           01154
01155
01156
01157
           /* Set time string... */
           jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01158
01159
01160
                    year, mon, day, hour, min);
01161
           /* Dump gnuplot file to pipe... */
01162
           if (!(in = fopen(ctl->atm_gpfile, "r")))
ERRMSG("Cannot open file!");
01163
01164
           while (fgets(line, LEN, in))
fprintf(out, "%s", line);
01165
01166
01167
           fclose(in);
01168
        }
01169
01170
        else {
01171
01172
          /* Write info... */
```

```
printf("Write atmospheric data: %s\n", filename);
01174
01175
          /* Create file... */
         if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");
01176
01177
01178
01179
01180
        /* Write header... */
01181
       fprintf(out,
                "# $1 = time [s]\n"
"# $2 = altitude [km]\n"
01182
01183
                "# \$3 = longitude [deg] \n" "# \$4 = latitude [deg] \n");
01184
       01185
01186
01187
       fprintf(out, "\n");
01188
01189
01190
        /* Write data... */
       01191
01192
01193
         for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01194
01195
01196
01197
01198
         fprintf(out, "\n");
01199
01200
01201
       /* Close file... */
01202
       fclose(out);
01203 }
```

Here is the call graph for this function:



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

Write CSI data.

Definition at line 1207 of file libtrac.c.

```
01211
01212
       static FILE *in, *out;
01214
01215
       static char line[LEN];
01216
       static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01217
01218
         rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01219
01220
       static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01221
        /* Init... */
01222
01223
        if (!init) {
01224
         init = 1;
01225
01226
          /\star Check quantity index for mass... \star/
01227
         if (ctl->qnt_m < 0)
           ERRMSG("Need quantity mass to analyze CSI!");
01228
01229
01230
          /* Open observation data file... */
01231
         printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
01232
          if (!(in = fopen(ctl->csi_obsfile, "r")))
```

```
01233
           ERRMSG("Cannot open file!");
01234
01235
          /* Create new file... */
         01236
01237
           ERRMSG("Cannot create file!");
01238
01239
01240
          /* Write header... */
01241
          fprintf(out,
01242
                  "# $1 = time [s] \n"
                  "# $2 = number of hits (cx)\n"
01243
                  "# $3 = number of misses (cy)\n"
01244
01245
                  "# $4 = number of false alarms (cz)\n"
01246
                  "# $5 = number of observations (cx + cy) n"
01247
                  "# $6 = number of forecasts (cx + cz)n"
01248
                  "# \$7 = bias (forecasts/observations) [\%] \n"
                  "# $8 = probability of detection (POD) [%%]\n" "# $9 = false alarm rate (FAR) [%%]\n" "# $10 = critical success index (CSI) [%%]\n\n");
01249
01250
01251
01252
       }
01253
01254
       /* Set time interval... */
       t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01255
01256
01257
01258
        /* Initialize grid cells... */
01259
        for (ix = 0; ix < ctl->csi_nx; ix++)
01260
         for (iy = 0; iy < ctl->csi_ny; iy++)
01261
            for (iz = 0; iz < ctl->csi_nz; iz++)
01262
             modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01263
01264
        /* Read data... */
01265
        while (fgets(line, LEN, in)) {
01266
          /* Read data... *,
01267
          if (sscanf(line, "%lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01268
01269
              5)
01270
            continue;
01271
01272
          /* Check time... */
01273
          if (rt < t0)
01274
           continue;
          if (rt > t1)
01275
01276
           break;
01277
01278
          /* Calculate indices... */
01279
          ix = (int) ((rlon - ctl->csi\_lon0))
                      / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01280
          iy = (int) ((rlat - ctl -> csi_lat0))
01281
01282
                      / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01283
         iz = (int) ((rz - ctl -> csi_z0)
01284
                      / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01285
         /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
01286
01287
             iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01288
            continue:
01290
01291
          /* Get mean observation index... */
01292
         obsmean[ix][iy][iz] += robs;
01293
         obscount[ix][iy][iz]++;
01294
01295
01296
        /* Analyze model data... */
01297
        for (ip = 0; ip < atm->np; ip++) {
01298
         01299
01300
01301
           continue;
01302
01303
          /* Get indices... */
01304
          ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
                      / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01305
         01306
01307
         01308
01309
01310
         /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
    iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01311
01312
01313
01314
            continue;
01315
01316
          /∗ Get total mass in grid cell... ∗/
01317
          modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01318
01319
```

```
/* Analyze all grid cells... */
         for (ix = 0; ix < ctl->csi_nx; ix++)
01321
01322
           for (iy = 0; iy < ctl->csi_ny; iy++)
             for (iz = 0; iz < ctl->csi_nz; iz++) {
01323
01324
01325
                /* Calculate mean observation index... */
               if (obscount[ix][iy][iz] > 0)
01326
                  obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01327
01328
01329
                /\star Calculate column density... \star/
               if (modmean[ix][iy][iz] > 0) {
01330
                 dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
01331
01332
01333
                 lat = ctl -> csi_lat0 + dlat * (iy + 0.5);
                 area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180. * cos(lat * M_PI / 180.);
01334
01335
                  modmean[ix][iy][iz] /= (1e6 * area);
01336
               }
01337
01338
01339
                /* Calculate CSI... */
01340
                if (obscount[ix][iy][iz] > 0) {
01341
                  if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01342
                      modmean[ix][iy][iz] >= ctl->csi_modmin)
01343
                    cx++;
01344
                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
                             modmean[ix][iy][iz] < ctl->csi_modmin)
01345
01346
                    cy++;
01347
                  else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
01348
                             modmean[ix][iy][iz] >= ctl->csi_modmin)
01349
                    cz++;
01350
01351
             }
01352
01353
         /* Write output... */
01354
        if (fmod(t, ctl->csi_dt_out) == 0) {
01355
           01356
01357
01358
                    t, cx, cy, cz, cx + cy, cx + cz,
                     t, cx, cy, cz, cx + cy, cx + cz,
(cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
(cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
(cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
(cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01359
01360
01361
01362
01363
01364
           /* Set counters to zero... */
           cx = cy = cz = 0;
01365
01366
01367
         /* Close file... */
01368
         if (t == ctl->t_stop)
01369
01370
           fclose(out);
01371 }
```

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

Write gridded data.

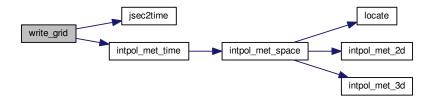
Definition at line 1375 of file libtrac.c.

```
01381
                  {
01382
01383
       FILE *in, *out;
01384
01385
       char line[LEN];
01386
01387
       static double grid m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01388
         area, rho air, press, temp, cd, mmr, t0, t1, r;
01389
01390
       static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01391
01392
        /* Check dimensions... */
       if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
01393
01394
         ERRMSG("Grid dimensions too large!");
01395
01396
       /* Check quantity index for mass... */
01397
       if (ctl->qnt_m < 0)
01398
         ERRMSG("Need quantity mass to write grid data!");
01399
01400
       /* Set time interval for output... */
01401
       t0 = t - 0.5 * ctl->dt_mod;
```

```
t1 = t + 0.5 * ct1->dt_mod;
01402
01403
01404
         /* 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;
01405
01406
01407
01408
01409
          /* Initialize grid... */
01410
         for (ix = 0; ix < ctl->grid_nx; ix++)
           for (iy = 0; iy < ctl->grid_ny; iy++)
  for (iz = 0; iz < ctl->grid_nz; iz++)
01411
01412
                grid_m[ix][iy][iz] = 0;
01413
01414
01415
         /* Average data... */
01416
         for (ip = 0; ip < atm->np; ip++)
           if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {</pre>
01417
01418
01419
              /* Get index... */
              ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
              iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
01421
01422
              iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01423
              /* Check indices... */
if (ix < 0 || ix >= ctl->grid_nx ||
    iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01424
01425
01426
01427
                continue:
01428
              /* Add mass... */
01429
              grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01430
01431
01432
         /* Check if gnuplot output is requested... */ if (ctl->grid_gpfile[0] != '-') {
01433
01434
01435
           /* Write info... */
printf("Plot grid data: %s.png\n", filename);
01436
01437
01438
01439
            /* Create gnuplot pipe... */
           if (!(out = popen("gnuplot", "w")))
01440
01441
             ERRMSG("Cannot create pipe to gnuplot!");
01442
           01443
01444
01445
01446
            /* Set time string... */
01447
            jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01448
           fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01449
                     year, mon, day, hour, min);
01450
01451
            /* Dump gnuplot file to pipe... */
           if (!(in = fopen(ctl->grid_gpfile, "r")))
    ERRMSG("Cannot open file!");
01452
01453
           while (fgets(line, LEN, in))
fprintf(out, "%s", line);
01454
01455
01456
           fclose(in);
01457
         }
01458
01459
01460
01461
            /* Write info... */
           printf("Write grid data: %s\n", filename);
01462
01463
01464
            /* Create file... */
01465
           if (!(out = fopen(filename, "w")))
01466
              ERRMSG("Cannot create file!");
01467
01468
         /* Write header... */
01469
01470
         fprintf(out,
                   "# $1 = time [s] \n"
01472
                   "# $2 = altitude [km] \n"
                   "# $3 = longitude [deg] \n"
01473
                   "# $4 = latitude [deg]\n"
01474
                   "# $5 = surface area [km^2]\n"
"# $6 = layer width [km]\n"
01475
01476
01477
                   "# $7 = temperature [K] \n"
                  "# $8 = column density [kg/m^2]\n"
"# $9 = mass mixing ratio [1]\n\n";
01478
01479
01480
01481
         /* Write data... */
         for (ix = 0; ix < ctl->grid_nx; ix++) {
01482
          if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01484
              fprintf(out, "\n");
01485
            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++)
01486
01487
01488
```

```
if (!ctl->grid_sparse
01490
                     || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01491
                   /* Set coordinates... */
01492
                   z = ctl->grid_z0 + dz * (iz + 0.5);
lon = ctl->grid_lon0 + dlon * (ix + 0.5);
01493
01494
01495
                   lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01496
01497
                   /\star Get pressure and temperature... \star/
01498
                   press = P(z);
                   01499
01500
01501
01502
                   /* Calculate surface area... */
                   area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
 * cos(lat * M_PI / 180.);
01503
01504
01505
                  /* Calculate column density... */
cd = grid_m[ix][iy][iz] / (le6 * area);
01506
01507
01508
                  /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01509
01510
01511
01512
01513
                    /* Write output... */
01514
                   fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g\n", t, z, lon, lat, area, dz, temp, cd, mmr);
01515
01516
01517
           }
01518
        }
01519
01520
          /* Close file... */
01521 fclose(out);
01522 }
```

Here is the call graph for this function:



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

Write profile data.

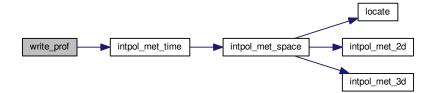
Definition at line 1526 of file libtrac.c.

```
01532
                   {
01533
01534
        static FILE *in, *out;
01535
        static char line[LEN];
01537
        static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
01538
         rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, mmr, h2o, o3;
01539
01540
01541
01542
        static int init, obscount[GX][GY], ip, ix, iy, iz;
01543
        /* Init... */
01544
        if (!init) {
01545
01546
          init = 1;
01547
01548
          /* Check quantity index for mass... */
```

```
if (ctl->qnt_m < 0)
01550
             ERRMSG("Need quantity mass!");
01551
           /* Check dimensions... */
if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
    ERRMSG("Grid dimensions too large!");
01552
01553
01554
01555
01556
            /* Open observation data file... */
01557
           printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01558
            if (!(in = fopen(ctl->prof_obsfile, "r")))
              ERRMSG("Cannot open file!");
01559
01560
           /* Create new file... */
01561
           printf("Write profile data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01562
01563
             ERRMSG("Cannot create file!");
01564
01565
01566
            /* Write header... */
01567
           fprintf(out,
                     "# $1
01568
                             = time [s]\n"
01569
                     "# $2
                            = altitude [km]\n"
                            = longitude [deg]\n"
01570
                     "# $3
                     "# $4 = latitude [deg]\n"
01571
                     "# $5
                            = pressure [hPa]\n"
01572
01573
                     "# $6 = temperature [K]\n"
01574
                     "# $7 = mass mixing ratio [1]\n"
01575
                     "# $8 = H20 volume mixing ratio [1]\n"
01576
                     "# $9 = 03 volume mixing ratio [1]\n"
                     "# $10 = mean BT index [K] \n");
01577
01578
01579
           /* Set grid box size... */
01580
           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;
01581
01582
01583
01584
         /* Set time interval... */
01585
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01587
01588
01589
         /* Initialize... */
         for (ix = 0; ix < ctl->prof_nx; ix++)
01590
          for (iy = 0; iy < ctl->prof_ny; iy++) {
01591
              obsmean[ix][iy] = 0;
01592
              obscount[ix][iy] = 0;
01593
01594
              tmean[ix][iy] = 0;
01595
              for (iz = 0; iz < ctl->prof_nz; iz++)
01596
                mass[ix][iy][iz] = 0;
01597
01598
01599
         /* Read data... */
01600
         while (fgets(line, LEN, in)) {
01601
           /* Read data... */
if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01602
01603
01604
             continue;
01605
01606
           /* Check time... */
01607
           if (rt < t0)</pre>
           continue;
if (rt > t1)
01608
01609
01610
             break;
01611
01612
            /* Calculate indices... */
           ix = (int) ((rlon - ctl->prof_lon0) / dlon);
iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01613
01614
01615
           /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01616
01617
             continue;
01619
01620
           /\star Get mean observation index... \star/
           obsmean[ix][iy] += robs;
tmean[ix][iy] += rt;
01621
01622
           obscount[ix][iy]++;
01623
01624
01625
01626
         /* Analyze model data... */
01627
         for (ip = 0; ip < atm->np; ip++) {
01628
01629
           /* Check time... */
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
01630
01631
             continue;
01632
           /* Get indices... */
01633
           ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
01634
01635
```

```
iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01638
             /* Check indices... */
            if (ix < 0 || ix >= ctl->prof_nx ||
   iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01639
01640
01641
               continue;
01642
01643
             /\star Get total mass in grid cell... \star/
01644
            mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01645
01646
01647
          /* Extract profiles... */
          for (ix = 0; ix < ctl->prof_nx; ix++)
  for (iy = 0; iy < ctl->prof_ny; iy++)
01648
01649
01650
               if (obscount[ix][iy] > 0) {
01651
                 /* Write output... */
fprintf(out, "\n");
01652
01653
01654
                  /* Loop over altitudes... */
01656
                  for (iz = 0; iz < ctl->prof_nz; iz++) {
01657
01658
                    /* Set coordinates... */
                    z = ctl->prof_z0 + dz * (iz + 0.5);
lon = ctl->prof_lon0 + dlon * (ix + 0.5);
01659
01660
                    lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01661
01662
01663
                    /* Get meteorological data... */
01664
                    press = P(z);
                    intpol_met_time(met0, met1, t, press, lon, lat,
    NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
01665
01666
01667
                    /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
01668
01669
                   area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
 * cos(lat * M_PI / 180.);
01670
01671
                    mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
01672
01673
01674
                     /* Write output... */
                    fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g \n", tmean[ix][iy] / obscount[ix][iy],
01675
01676
                              z, lon, lat, press, temp, mmr, h2o, o3,
obsmean[ix][iy] / obscount[ix][iy]);
01677
01678
01679
01680
01681
01682
          /\star Close file... \star/
          if (t == ctl->t_stop)
01683
01684
            fclose(out);
01685 }
```

Here is the call graph for this function:



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

Write station data.

Definition at line 1689 of file libtrac.c.

```
{
01694
01695
        static FILE *out;
01696
01697
         static double rmax2, t0, t1, x0[3], x1[3];
01698
01699
        static int init, ip, iq;
01700
01701
         /* Init... */
01702
         if (!init) {
01703
           init = 1;
01704
01705
           /* Write info... */
01706
           printf("Write station data: %s\n", filename);
01707
01708
           /\star Create new file... \star/
           if (!(out = fopen(filename, "w")))
01709
             ERRMSG("Cannot create file!");
01710
01711
01712
           /* Write header... */
01713
           fprintf(out,
                     "# $1 = time [s]\n"
"# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01714
01715
01716
           for (iq = 0; iq < ctl->nq; iq+)
fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
ctl->qnt_name[iq], ctl->qnt_unit[iq]);
fprintf(out, "\n");
01717
01718
01719
01720
01721
           /\star Set geolocation and search radius... \star/
01722
01723
           geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
01724
           rmax2 = gsl_pow_2(ctl->stat_r);
01725
01726
        /* Set time interval for output... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01727
01728
01729
01730
01731
         /* Loop over air parcels... */
01732
         for (ip = 0; ip < atm->np; ip++) {
01733
01734
           /* Check time... */
01735
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
01736
             continue;
01737
01738
           /\star Check station flag... \star/
01739
           if (ctl->qnt_stat >= 0)
01740
             if (atm->q[ctl->qnt_stat][ip])
01741
                continue:
01742
01743
           /* Get Cartesian coordinates... */
01744
           geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
01745
01746
           /\star Check horizontal distance... \star/
01747
           if (DIST2(x0, x1) > rmax2)
01748
             continue;
01749
01750
           /* Set station flag... */
01751
           if (ctl->qnt_stat >= 0)
01752
             atm->q[ctl->qnt_stat][ip] = 1;
01753
           /* Write data... */
fprintf(out, "%.2f %g %g %g",
01754
01755
01756
                    atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
           for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01757
01758
01759
01760
01761
           fprintf(out, "\n");
01762
01763
01764
         /* Close file... */
         if (t == ctl->t_stop)
01765
01766
           fclose(out);
01767 }
```

5.12 libtrac.c 65

Here is the call graph for this function:



5.12 libtrac.c

```
00001 /*
00002
      This file is part of MPTRAC.
00003
00004
      \ensuremath{\mathsf{MPTRAC}} is free software: you can redistribute it and/or modify
00005
      it under the terms of the GNU General Public License as published by
00006
      the Free Software Foundation, either version 3 of the License, or
00007
      (at your option) any later version.
00008
00009
      \ensuremath{\mathsf{MPTRAC}} is distributed in the hope that it will be useful,
      but WITHOUT ANY WARRANTY; without even the implied warranty of
00010
00011
      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
      GNU General Public License for more details.
00013
00014
      You should have received a copy of the GNU General Public License
00015
      along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>.
00016
      Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00017
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;
      *lon = atan2(x[1], x[0]) * 180 / M_PI;
00039
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
       /* Avoid singularity at poles...
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 }
00098
00100
00101 void geo2cart(
00102
      double z,
00103
       double lon,
      double lat,
00104
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,
00121
       met_t * met0,
00122
      met_t * met1) {
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(filename, met0);
00134
00135
        get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
     dt met, filename);
00136
        read met(filename, met1);
00137
00138
00139
       /\star Read new data for forward trajectories... \star/
00140
       if (t > met1->time && ctl->direction == 1) {
       memcpy(met0, met1, sizeof(met_t));
get_met_help(t, 1, metbase, ctl->dt_met, filename);
00141
00142
00143
        read_met(filename, met1);
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);
00148
00149
00150
         read_met(filename, met0);
00151
00152 }
00153
```

5.12 libtrac.c 67

```
00155
00156 void get_met_help(
00157
       double t,
00158
       int direct,
00159
       char *metbase,
00160
       double dt met.
       char *filename) {
00161
00162
00163
       double t6, r;
00164
00165
       int year, mon, day, hour, min, sec;
00166
00167
       /* Round time to fixed intervals... */
00168
       if (direct == -1)
00169
         t6 = floor(t / dt_met) * dt_met;
00170
         t6 = ceil(t / dt_met) * dt_met;
00171
00172
00173
       /* Decode time... */
00174
       jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00175
        /* Set filename... */
00176
       sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", metbase, year, mon, day, hour);
00177
00178 }
00179
00181
00182 void intpol_met_2d(
00183
       double array[EX][EY],
00184
       int ix,
       int iy,
00185
00186
       double wx,
00187
       double wy,
00188
       double *var) {
00189
       double aux00, aux01, aux10, aux11;
00190
00191
00192
       /* Set variables... */
00193
       aux00 = array[ix][iy];
       aux01 = array[ix][iy + 1];
aux10 = array[ix + 1][iy];
00194
00195
       aux11 = array[ix + 1][iy + 1];
00196
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,
00209
       int ix,
00210
       int iy,
double wp,
00211
00212
       double wx,
       double wy,
00213
00214
       double *var) {
00215
00216
       double aux00, aux01, aux10, aux11;
00217
00218
       /* Interpolate vertically... */
00219
       aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00220
         + array[ix][iy][ip + 1];
00221
       \verb"aux01 = \verb"wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1]")
       + array[ix][iy + 1][ip + 1];
aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
00222
00223
         + array[ix + 1][iy][ip + 1];
00224
00225
       aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
00226
         + array[ix + 1][iy + 1][ip + 1];
00227
00228
       /* Interpolate horizontally... */
       aux00 = wy * (aux00 - aux01) + aux01;

aux11 = wy * (aux10 - aux11) + aux11;
00229
00230
00231
       *var = wx * (aux00 - aux11) + aux11;
00232 }
00233
00235
00236 void intpol_met_space(
00237
       met_t * met,
       double p,
00238
00239
       double lon,
00240
       double lat,
00241
       double *ps,
```

```
00242
        double *t,
        double *u,
00243
        double *v,
00244
00245
        double *w,
00246
        double *h2o.
00247
        double *o3) {
00248
00249
        double wp, wx, wy;
00250
00251
        int ip, ix, iy;
00252
       /* Check longitude... */
00253
00254
        if (lon < 0)</pre>
00255
          lon += 360;
00256
00257
        /\star Get indices... \star/
00258
       ip = locate(met->p, met->np, p);
        ix = locate(met->lon, met->nx, lon);
iy = locate(met->lat, met->ny, lat);
00259
00260
00261
00262
       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... */
00268
        if (ps != NULL)
00269
          intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
00270
        if (t != NULL)
         intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
00271
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)
          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,
00288
        met t * met1
00289
        double ts,
        double p, double lon,
00290
00291
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 h200, h201, o30, o31, ps0, ps1, t0, t1, u0, u1, v0, v1, w0, w1, wt;
00302
00303
        /* Spatial interpolation... */
        00304
00305
                          t == NULL ? NULL : &t0,
00306
00307
                          u == NULL ? NULL : &u0,
                          v == NULL ? NULL : &v0,
00308
                          w == NULL ? NULL : &w0,
00309
                          h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00310
        00311
00312
                          t == NULL ? NULL : &t1,
00313
                          u == NULL ? NULL : &u1,
00314
                          v == NULL ? NULL : &v1,
00315
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)
00324
          *ps = wt * (ps0 - ps1) + ps1;
        if (t != NULL)
00325
00326
         *t = wt * (t0 - t1) + t1;
        if (u != NULL)
00327
00328
         *u = wt * (u0 - u1) + u1;
```

5.12 libtrac.c 69

```
if (v != NULL)
00329
       *v = wt * (v0 - v1) + v1;
if (w != NULL)
00330
00331
       *W = Wt * (w0 - w1) + w1;

if (h2o != NULL)

*h2o = wt * (h2o0 - h2o1) + h2o1;

if (o3 != NULL)
00332
00333
00334
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;
00356
       t0.tm\_mon = 0;
00357
       t0.tm_mday = 1;
       t0.tm_hour = 0;
00358
00359
       t0.tm min = 0;
00360
       t0.tm\_sec = 0;
00361
00362
        jsec0 = (time_t) jsec + timegm(&t0);
00363
       t1 = gmtime(\&jsec0);
00364
00365
       *year = t1->tm_year + 1900;
*mon = t1->tm_mon + 1;
00366
00367
       *day = t1->tm_mday;
00368
       *hour = t1->tm_hour;
       *min = t1->tm_min;
*sec = t1->tm_sec;
00369
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;
00384
       ihi = n - 1;
00385
       i = (ihi + ilo) >> 1;
00386
00387
       if (xx[i] < xx[i + 1])
         while (ihi > ilo + 1) {
i = (ihi + ilo) >> 1;
00388
00389
           if (xx[i] > x)
00390
             ihi = i;
00391
00392
           else
00393
             ilo = i;
00394
       } else
         while (ihi > ilo + 1) {
  i = (ihi + ilo) >> 1;
00395
00396
           <u>if</u> (xx[i] <= x)
00397
00398
             ihi = i;
00399
           else
00400
             ilo = i;
00401
         }
00402
00403
       return ilo;
00404 }
00405
00407
00408 void read atm(
      const char *filename,
ctl_t * ctl,
00409
00410
00411
       atm_t * atm) {
00412
00413
       FILE *in;
00414
00415
       char line[LEN], *tok;
```

```
00416
00417
         int iq;
00418
00419
         /* Init... */
        atm->np = 0;
00420
00421
00422
         /* Write info... */
00423
         printf("Read atmospheric data: sn', 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
             ERRMSG("Too many data points!");
00445
00446
00447
00448
         /* Close file... */
00449
         fclose(in);
00450
00451
         /* Check number of points... */
00452
         if (atm->np < 1)
           ERRMSG("Can not read any data!");
00454 }
00455
00457
00458 void read_ctl(
00459
        const char *filename,
         int argc,
00461
         char *argv[],
00462
        ctl_t * ctl) {
00463
00464
        int ia:
00465
00466
         /* Write info... */
00467
         printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
00468
                  "(executable: %s | compiled: %s, %s)\n\n",
00469
                 argv[0], __DATE__, __TIME__);
00470
00471
         /* Initialize quantity indices... */
00472
         ctl->qnt_m = -1;
00473
         ctl->qnt_r = -1;
00474
         ctl->qnt_rho = -1;
         ctl->qnt_ps = -1;
00475
         ct1->qnt_t = -1;
00476
         ctl->qnt_u = -1;
00477
00478
         ctl->qnt_v = -1;
00479
         ctl->qnt_w = -1;
00480
         ct1->qnt_h2o = -1;
00481
         ctl->qnt_o3 = -1;
00482
         ctl->qnt\_theta = -1;
         ctl->qnt_stat = -1;
00483
00484
00485
         /* Read quantities... */
00486
         ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
         for (iq = 0; iq < ctl->nq; iq++) {
00487
00488
           /* 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",
00489
00490
00491
00492
                      ctl->qnt_format[iq]);
00493
           /* Try to identify quantity... */
if (strcmp(ctl->qnt_name[iq], "m") == 0) {
00494
00495
00496
             ctl->qnt_m = iq;
             sprintf(ctl->qnt_unit[iq], "kg");
00497
00498
           } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
             ctl->qnt_r = iq;
00499
             sprintf(ctl->qnt_unit[iq], "m");
00500
           } else if (strcmp(ct1->qnt_name[iq], "rho") == 0) {
ct1->qnt_rho = iq;
00501
00502
```

```
sprintf(ctl->qnt_unit[iq], "kg/m^3");
           } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
00504
00505
             ctl->qnt_ps = iq;
00506
             sprintf(ctl->qnt_unit[iq], "hPa");
           } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
  ctl->qnt_t = iq;
00507
00508
             sprintf(ctl->qnt_unit[iq], "K");
           } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
00510
            ctl->qnt_u = iq;
00511
00512
             sprintf(ctl->qnt_unit[iq], "m/s");
           } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
00513
00514
             ctl->qnt_v = iq;
sprintf(ctl->qnt_unit[iq], "m/s");
00515
00516
           } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
00517
             ctl->qnt_w = iq;
           sprintf(ctl->qnt_unit[iq], "hPa/s");
} else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
ctl->qnt_h2o = iq;
00518
00519
00520
             sprintf(ctl->qnt_unit[iq], "1");
00522
           } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
            ctl->qnt_o3 = iq;
00523
00524
             sprintf(ctl->qnt_unit[iq], "1");
           } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
00525
             ctl->qnt_theta = iq;
00526
00527
             sprintf(ctl->qnt_unit[iq], "K");
           } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
00528
             ctl->qnt_stat = iq;
00529
00530
             sprintf(ctl->qnt_unit[iq], "-");
00531
00532
             scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00533
00534
00535
         /\star Time steps of simulation... \star/
00536
         ctl->direction =
         (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL); if (ctl->direction != -1 && ctl->direction != 1)
00537
00538
          ERRMSG("Set DIRECTION to -1 or 1!");
00539
         ctl->t_start =
00541
           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);
00542
00543
00544
00545
         /* Meteorological data..
00546
         ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
00547
00548
         /* Isosurface parameters... */
00549
         ctl->isosurf
         = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00550
00551
00552
         /* Diffusion parameters... */
00554
        ctl->turb_dx_trop
00555
           = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00556
         ctl->turb dx strat
           = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00557
00558
         ctl->turb dz trop
            = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00560
         ctl->turb dz strat
00561
           = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00562
         ctl->turb meso
           scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00563
00564
00565
         /* Life time of particles... */
        ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL); ctl->tdec_strat =
00566
00567
00568
           scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00569
        /* Output of atmospheric data... */
scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
00570
00571
      atm_basename);
00572 scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00573
         ctl->atm_dt_out =
00574
           scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00575
00576
        /* Output of CSI data... */
         scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
00578 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",
00579
00580
00581
                   ctl->csi obsfile);
00582
        ctl->csi_obsmin =
           scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00583
00584
        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);
00585
00586
00587
```

```
ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
00589
        ctl->csi lon0 =
          scan_ctl(filename, argc, argv, "CSI_LONO", -1, "-180", NULL);
00590
        ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
00591
00592
        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);
00593
00594
00595
00596
        ctl->csi_ny =
00597
           (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
00598
        /* Output of grid data... */
00599
        00600
00601
00602
        scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
grid_gpfile);
00603 ct1=>~***
        ctl->grid_dt_out =
00604
          scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
        ctl->grid_sparse =
00605
00606
          (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
        ctl-ygrid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL); ctl-ygrid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
00607
00608
00609
        ctl->grid nz =
           (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00610
00611
        ctl->grid_lon0 =
00612
          scan_ctl(filename, argc, argv, "GRID_LONO", -1, "-180", NULL);
00613
         ctl->grid_lon1
00614
          scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
00615
        ctl->grid nx =
00616
          (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00617
        ctl->grid lat0 =
00618
           scan_ctl(filename, argc, argv, "GRID_LATO", -1, "-90", NULL);
00619
00620
          scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
        ctl->grid_ny =
00621
           (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
00622
00623
00624
        /* Output of profile data... */
        scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
00625
00626
                  ctl->prof_basename);
        scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
00627
      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);
00628
00629
00630
        ctl->prof_nz =
00631
           (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
00632
        ctl->prof_lon0 =
          scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
00633
00634
        ctl->prof lon1 =
          scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
00635
00636
        ctl->prof_nx =
           (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00637
        ctl->prof_lat0 =
00638
00639
          scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
00640
        ctl->prof lat1 =
00641
          scan ctl(filename, argc, argv, "PROF LAT1", -1, "90", NULL);
        ctl->prof_ny =
00642
          (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00643
00644
00645
        /* Output of station data... */
        00646
00647
        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);
00648
00649
00650
00651 }
00652
00654
00655 void read_met(
00656
        char *filename,
00657
        met_t * met) {
00658
00659
        char tstr[10]:
00660
00661
        static float help[EX * EY];
00662
00663
        int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00664
00665
        size t np, nx, nv;
00666
         /* Write info... */
00667
        printf("Read meteorological data: %s\n", filename);
00668
00669
        /* Get time from filename... */ sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00670
00671
00672
        vear = atoi(tstr);
```

```
sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00674
         mon = atoi(tstr);
00675
         sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00676
         day = atoi(tstr);
         sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00677
00678
         hour = atoi(tstr);
00679
         time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00680
         /* Open netCDF file... */
00681
00682
         NC(nc_open(filename, NC_NOWRITE, &ncid));
00683
00684
         /* Get dimensions... */
NC(nc_inq_dimid(ncid, "lon", &dimid));
00685
00686
         NC(nc_inq_dimlen(ncid, dimid, &nx));
00687
         if (nx > EX)
00688
           ERRMSG("Too many longitudes!");
00689
         NC(nc_inq_dimid(ncid, "lat", &dimid));
NC(nc_inq_dimlen(ncid, dimid, &ny));
00690
00691
00692
             (ny > EY)
00693
           ERRMSG("Too many latitudes!");
00694
         NC(nc_inq_dimid(ncid, "lev", &dimid));
00695
00696
         NC(nc_inq_dimlen(ncid, dimid, &np));
00697
         if (np > EP)
           ERRMSG("Too many pressure levels!");
00698
00699
00700
         /* Store dimensions... */
        met->np = (int) np;

met->nx = (int) nx;
00701
00702
00703
         met->ny = (int) ny;
00704
00705
         /* Read geolocations... */
NC(nc_inq_varid(ncid, "lev", &varid));
00706
00707
         NC(nc_get_var_double(ncid, varid, met->p));
00708
         NC(nc_inq_varid(ncid, "lon", &varid));
NC(nc_get_var_double(ncid, varid, met->lon));
00709
00710
00711
00712
         NC(nc_inq_varid(ncid, "lat", &varid));
00713
         NC(nc_get_var_double(ncid, varid, met->lat));
00714
00715
         /* Check and convert pressure levels... */
         for (ip = 0; ip < met->np; ip++) {
  if (ip > 0 && met->p[ip - 1] > met->p[ip])
00716
00717
00718
             ERRMSG("Pressure levels must be descending!");
00719
           met->p[ip] /= 100.;
00720
00721
00722
         /* Read surface pressure... */
             (nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00724
           NC(nc_get_var_float(ncid, varid, help));
           for (iy = 0; iy < met->ny; iy++)
  for (ix = 0; ix < met->nx; ix++)
00725
00726
00727
               met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00728
         } else {
00729
           for (ix = 0; ix < met->nx; ix++)
00730
              for (iy = 0; iy < met->ny; iy++)
00731
                met \rightarrow ps[ix][iy] = met \rightarrow p[0];
00732
00733
         /* Read meteorological data... */
read_met_help(ncid, "t", "T", met, met->np, met->t, 1.0);
read_met_help(ncid, "u", "U", met, met->np, met->u, 1.0);
read_met_help(ncid, "v", "V", met, met->np, met->v, 1.0);
00734
00735
00736
00737
         read_met_help(ncid, "w", "W", met, met->np, met->w, 0.01f);
read_met_help(ncid, "q", "Q", met, met->np, met->h2o, 1.608f);
read_met_help(ncid, "o3", "03", met, met->np, met->o3, 0.602f);
00738
00739
00740
00741
00742
         /\star Extrapolate data for lower boundary... \star/
00743
         read_met_extrapolate(met);
00744
00745
         /\star Copy data to obtain periodic boundary conditions... \star/
00746
         read_met_periodic(met);
00747
00748
         /* Close file...
         NC(nc_close(ncid));
00749
00750 }
00751
00753
00754 void read_met_extrapolate(
00755
        met t * met) {
00756
00757
        int ip, ip0, ix, iy;
00758
00759
        /* Loop over columns... */
```

```
for (ix = 0; ix < met->nx; ix++)
00761
           for (iy = 0; iy < met->ny; iy++) {
00762
00763
              /* Find lowest valid data point... */
00764
              for (ip0 = met->np - 1; ip0 >= 0; ip0--)
if (!gsl_finite(met->t[ix][iy][ip0])
00765
00766
                     || !gsl_finite(met->u[ix][iy][ip0])
00767
                     || !gsl_finite(met->v[ix][iy][ip0])
00768
                     || !gsl_finite(met->w[ix][iy][ip0]))
00769
00770
00771
              /* Extrapolate... */
             for (ip = ip0; ip >= 0; ip--) {
  met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
00772
00773
00774
                met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
               met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
00775
00776
00777
                met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
00778
               met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00779
00780
00781 }
00782
00783 /
        00784
00785 void read_met_help(
00786
         int ncid,
00787
         char *varname,
00788
         char *varname2,
00789
         met_t * met,
00790
         int np,
00791
         float dest[EX][EY][EP],
00792
         float scl) {
00793
00794
         static float help[EX * EY * EP];
00795
00796
         int ip, ix, iv, n = 0, varid;
00797
00798
         /* Check if variable exists... */
00799
         if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00800
           if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00801
             return;
00802
00803
         /* Read data... */
         NC(nc_get_var_float(ncid, varid, help));
00804
00805
00806
         /* Copy and check data... */
00807
         for (ip = 0; ip < np; ip++)</pre>
00808
           for (iy = 0; iy < met->ny; iy++)
             for (iy = 0; iy < met->ny; iy++)
for (ix = 0; ix < met->nx; ix++) {
  dest[ix][iy][ip] = scl * help[n++];
  if (dest[ix][iy][ip] < -1e10 || dest[ix][iy][ip] > 1e10)
  dest[ix][iy][ip] = GSL_NAN;
00809
00810
00811
00812
00813
00814 }
00815
00817
00818 void read_met_periodic(
00819
        met_t * met) {
00820
00821
         int ip, iy;
00822
00823
         /* Check longitudes... */
         if (fabs(met->lon[met->nx - 1] - met->lon[0] - 360) < 0.01)
00824
00825
           return;
00826
00827
         /* Increase longitude counter... */
00828
         if ((++met->nx) > EX)
00829
           ERRMSG("Cannot create periodic boundary conditions!");
00830
00831
         /* Set longitude... */
00832
        met - lon[met - nx - 1] = met - lon[met - nx - 2] + met - lon[1] - met - lon[1]
      lon[0];
00833
00834
         /* Loop over latitudes and pressure levels... */
00835
         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];
00836
00837
00838
             met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
00839
             met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
00840
00841
00842
             met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
             met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
00843
00844
00845 }
```

```
00848
00849 double scan_ctl(
00850
        const char *filename,
00851
        int argc.
        char *argv[],
00853
        const char *varname,
00854
        int arridx,
00855
        const char *defvalue,
00856
        char *value) {
00857
00858
        FILE *in = NULL;
00859
00860
        char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
00861
         msg[LEN], rvarname[LEN], rval[LEN];
00862
00863
        int contain = 0, i;
00864
00865
        /* Open file... */
        if (filename[strlen(filename) - 1] != '-')
if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");
00866
00867
00868
00869
00870
        /* Set full variable name... */
00871
        if (arridx >= 0) {
          sprintf(fullname1, "%s[%d]", varname, arridx);
00872
          sprintf(fullname2, "%s[*]", varname);
00873
00874
         sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
00875
00876
00877
00878
00879
        /* Read data... */
00880
        if (in != NULL)
          while (fgets(line, LEN, in))
  if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
  if (strcasecmp(rvarname, fullname1) == 0 ||
00881
00882
00884
                   strcasecmp(rvarname, fullname2) == 0) {
00885
                 contain = 1;
00886
                break;
00887
              }
        for (i = 1; i < argc - 1; i++)</pre>
00888
         if (strcasecmp(argv[i], fullname1) == 0 ||
    strcasecmp(argv[i], fullname2) == 0) {
    sprintf(rval, "%s", argv[i + 1]);
00889
00890
00891
00892
            contain = 1;
00893
            break;
         }
00894
00895
00896
        /* Close file... */
00897
        if (in != NULL)
00898
          fclose(in);
00899
00900
        /* Check for missing variables... */
00901
        if (!contain) {
00902
         if (strlen(defvalue) > 0)
00903
            sprintf(rval, "%s", defvalue);
00904
            sprintf(msg, "Missing variable s!\n", fullname1);
00905
00906
            ERRMSG(msg);
00907
00908
00909
00910
        /* Write info... */
00911
        printf("%s = %s\n", fullname1, rval);
00912
00913
        /* Return values... */
00914
        if (value != NULL)
         sprintf(value, "%s", rval);
00916
        return atof(rval);
00917 }
00918
00920
00921 void time2jsec(
00922
        int year,
00923
        int mon,
00924
        int day,
00925
        int hour.
00926
        int min,
00927
        int sec,
00928
        double remain,
00929
        double *jsec) {
00930
00931
        struct tm t0, t1;
00932
```

```
t0.tm_year = 100;
00934
                 t0.tm\_mon = 0;
00935
                 t0.tm_mday = 1;
                t0.tm_hour = 0;
00936
                 t0.tm_min = 0;
00937
00938
                 t0.tm\_sec = 0;
00939
00940
                 t1.tm_year = year - 1900;
00941
                t1.tm_mon = mon - 1;
00942
                t1.tm mday = day;
                t1.tm_hour = hour;
00943
00944
                t1.tm min = min;
00945
                t1.tm_sec = sec;
00946
00947
                 *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
00948 }
00949
00951
00952 void timer(
00953
               const char *name,
00954
                int id.
00955
                int mode) {
00956
00957
                static double starttime[NTIMER], runtime[NTIMER];
00958
00959
00960
                if (id < 0 || id >= NTIMER)
00961
                    ERRMSG("Too many timers!");
00962
00963
                /* Start timer... */
00964
                if (mode == 1) {
00965
                  if (starttime[id] <= 0)</pre>
00966
                        starttime[id] = omp_get_wtime();
00967
                         ERRMSG("Timer already started!");
00968
00969
                }
00970
00971
                 /* Stop timer... */
00972
                 else if (mode == 2) {
00973
                    if (starttime[id] > 0) {
00974
                         runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
00975
                         starttime[id] = -1;
00976
                    } else
00977
                         ERRMSG("Timer not started!");
00978
00979
00980
                /* Print timer... */
00981
                else if (mode == 3)
                   printf("%s = %g s\n", name, runtime[id]);
00982
00983 }
00984
00986
00987 double tropopause(
00988
                double t,
                double lat)
00990
00991
                static double doys[12]
                = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00992
00993
00994
                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,
00995
00996
00997
                     -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,
00998
00999
01000
01001
                     75, 77.5, 80, 82.5, 85, 87.5, 90
01002
01003
01004
                 static double tps[12][73]
                    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, 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, 12
01005
01006
01007
01008
01009
                 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}, 275.3, 275.6, 275.4, 274.1, 273.5}, 275.3, 275.6, 275.4, 274.1, 273.5}, 275.3, 275.6, 275.4, 274.1, 273.5}, 275.3, 275.6, 275.4, 274.1, 273.5}, 275.3, 275.6, 275.4, 274.1, 273.5}, 275.3, 275.6, 275.4, 274.1, 273.5}, 275.3, 275.6, 275.4, 274.1, 273.5}, 275.3, 275.6, 275.4, 274.1, 273.5}, 275.3, 275.6, 275.4, 274.1, 273.5}, 275.3, 275.6, 275.4, 274.1, 273.5}, 275.3, 275.6, 275.4, 274.1, 273.5}, 275.3, 275.6, 275.4, 274.1, 273.5}, 275.3, 275.6, 275.4, 274.1, 273.5}, 275.3, 275.6, 275.4, 274.1, 273.5}, 275.3, 275.6, 275.4, 275.4, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 275.2, 
01010
01011
01012
01013
                   150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42, 98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
01015
01016
                   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,
01017
01018
01019
```

```
287.5, 286.2, 285.8},
          {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
01021
01022
           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,
01023
01024
           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,
01025
           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,
01027
01028
           304.3, 304.9, 306, 306.6, 306.2, 306},
01029
          {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,
01030
           195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
01031
           102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
01032
           99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
01033
01034
           148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
           263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4, 315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
01035
01036
          {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,
01037
           205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
           101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
01040
01041
           102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
01042
           165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
           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},
01043
01044
          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,
01046
01047
           228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
01048
           105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
01049
           106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
           127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245, 251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
01050
           308.5, 312.2, 313.1, 313.3},
01052
01053
          {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
           187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6, 235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
01054
01055
           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,
01056
01058
           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
01059
01060
           275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
          {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01061
           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,
01062
01063
           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,
01064
01065
01066
           120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
01067
           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}, {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
01068
01069
           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,
01071
01072
           114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
           110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6, 114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
01073
01074
           203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
01075
           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,
01077
01078
           215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
01079
           237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
           111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7, 106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
01080
01081
01082
           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, 279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
01084
           305.1},
01085
01086
          {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
           253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3, 223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
01087
01088
           108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5,
           102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
01090
01091
           109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
           241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6, 286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6}
01092
01093
          {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,
01094
           175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
01096
           100.4, 99.96, 99.67, 99.37, 99.32, 99.31, 99.46, 99.77, 100.2, 100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160, 186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278, 280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
01097
01098
01099
01100
           281.7, 281.1, 281.2}
01102
01103
01104
          double doy, p0, p1, pt;
01105
          int imon, ilat;
01106
```

```
01107
       /* Get day of year... */
doy = fmod(t / 86400., 365.25);
while (doy < 0)
01108
01109
01110
         doy += 365.25;
01111
01112
01113
       /* Get indices... */
01114
       imon = locate(doys, 12, doy);
01115
       ilat = locate(lats, 73, lat);
01116
01117
       /* Get tropopause pressure... */
       01118
01119
01120
01121
01122
       pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01123
01124
       /* Return tropopause pressure... */
01125
       return pt;
01126 }
01127
01129
01130 void write atm(
01131
       const char *filename,
       ctl_t * ctl,
01132
01133
       atm_t * atm,
01134
       double t) {
01135
01136
       FILE *in. *out;
01137
01138
       char line[LEN];
01139
01140
       double r;
01141
01142
       int ip, iq, year, mon, day, hour, min, sec;
01143
01144
       /* Check if gnuplot output is requested... */
01145
       if (ctl->atm_gpfile[0] != '-') {
01146
01147
         /* Write info... */
         printf("Plot atmospheric data: s.png\n", filename);
01148
01149
         /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
01150
01151
01152
           ERRMSG("Cannot create pipe to gnuplot!");
01153
         /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
01154
01155
01156
01157
         /* Set time string... */
         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01158
01159
01160
                 year, mon, day, hour, min);
01161
         /* Dump gnuplot file to pipe... */
01162
         if (!(in = fopen(ctl->atm_gpfile, "r")))
01163
01164
           ERRMSG("Cannot open file!");
         while (fgets(line, LEN, in))
fprintf(out, "%s", line);
01165
01166
01167
         fclose(in);
01168
01169
01170
01171
01172
         /\star Write info... \star/
01173
         printf("Write atmospheric data: %s\n", filename);
01174
01175
          /* Create file... */
         if (!(out = fopen(filename, "w")))
01176
01177
           ERRMSG("Cannot create file!");
01178
01179
        /* Write header... */
01180
01181
       fprintf(out,
01182
               "# $1 = time [s] \n"
01183
               "# $2 = altitude [km] \n"
01184
               "# $3 = longitude [deg] \n" "# <math>$4 = latitude [deg] \n");
       01185
01186
01187
       fprintf(out, "\n");
01188
01189
01190
        /* Write data... */
       01191
01192
01193
```

```
for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01195
01196
01197
          fprintf(out, "\n");
01198
01199
01200
01201
        /* Close file... */
01202
       fclose(out);
01203 }
01204
01206
01207 void write_csi(
01208
       const char *filename,
        ctl_t * ctl,
atm t * atm,
01209
01210
01211
        double t) {
01212
01213
        static FILE *in, *out;
01214
01215
       static char line[LEN];
01216
        static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01217
01218
          rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01219
        static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01220
01221
        /* Init... */
01222
        if (!init) {
01223
01224
          init = 1:
01225
01226
          /\star Check quantity index for mass... \star/
01227
          if (ctl->qnt_m < 0)
01228
            ERRMSG("Need quantity mass to analyze CSI!");
01229
01230
          /* Open observation data file... */
          printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
01231
01232
             (!(in = fopen(ctl->csi_obsfile, "r")))
01233
           ERRMSG("Cannot open file!");
01234
01235
          /* Create new file... */
          printf("Write CSI data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01236
01237
01238
            ERRMSG("Cannot create file!");
01239
01240
          /* Write header... */
          fprintf(out,
    "# $1 = time [s]\n"
01241
01242
                   "# $2 = number of hits (cx)\n"
01243
01244
                   "# $3 = number of misses (cy) \n"
01245
                   "# $4 = number of false alarms (cz)\n"
01246
                   "# $5 = number of observations (cx + cy) \n"
01247
                   "# $6 = number of forecasts (cx + cz) n"
                   "# $7 = bias (forecasts/observations) [%%]\n"
01248
                   "# $8 = probability of detection (POD) [%%]\n" # $9 = false alarm rate (FAR) [%%]\n"
01249
01250
01251
                   "# $10 = critical success index (CSI) [%%]\n\n");
01252
01253
        /\star Set time interval... \star/
01254
01255
        t0 = t - 0.5 * ctl->dt_mod;
        t1 = t + 0.5 * ctl->dt_mod;
01257
01258
        /* Initialize grid cells... */
01259
        for (ix = 0; ix < ctl->csi_nx; ix++)
01260
          for (iy = 0; iy < ctl->csi_ny; iy++)
  for (iz = 0; iz < ctl->csi_nz; iz++)
01261
01262
              modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01263
01264
        /* Read data... */
01265
        while (fgets(line, LEN, in)) {
01266
          /* Read data... */
if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01267
01268
01269
01270
            continue;
01271
01272
          /* Check time... */
01273
          if (rt < t.0)
01274
            continue;
01275
          if (rt > t1)
            break;
01276
01277
01278
          /* Calculate indices... */
          ix = (int) ((rlon - ctl->csi_lon0))
01279
01280
                       / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
```

```
iy = (int) ((rlat - ctl->csi_lat0)
                         / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01282
01283
           iz = (int) ((rz - ctl->csi_z0)
01284
                        / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01285
          /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
01286
01287
01288
               iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
             continue;
01289
01290
01291
           /* Get mean observation index... */
01292
          obsmean[ix][iv][iz] += robs;
01293
          obscount[ix][iy][iz]++;
01294
01295
01296
         /* Analyze model data... */
01297
        for (ip = 0; ip < atm->np; ip++) {
01298
           /* Check time... */
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
01300
01301
01302
01303
           /* Get indices... */
          ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01304
01305
                         / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
           iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
01306
01307
                          (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
          01308
01309
01310
01311
           /* Check indices... */
          if (ix < 0 || ix >= ctl->csi_nx ||
    iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01312
01313
01314
             continue;
01315
           /* Get total mass in grid cell... */
01316
          modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01317
01318
01319
01320
         /* Analyze all grid cells... */
        for (ix = 0; ix < ctl->csi_nx; ix++)
  for (iy = 0; iy < ctl->csi_ny; iy++)
01321
01322
             for (iz = 0; iz < ctl->csi_nz; iz++) {
01323
01324
01325
                /\star Calculate mean observation index... \star/
01326
               if (obscount[ix][iy][iz] > 0)
01327
                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01328
01329
                /* Calculate column density... */
               if (modmean[ix][iy][iz] > 0) {
  dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
01330
01332
                  dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
01333
                  lat = ctl->csi_lat0 + dlat * (iy + 0.5);
                 area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
  * cos(lat * M_PI / 180.);
modmean[ix][iy][iz] /= (le6 * area);
01334
01335
01336
01338
01339
                /* Calculate CSI...
01340
               if (obscount[ix][iy][iz] > 0) {
                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01341
                      modmean[ix][iy][iz] >= ctl->csi_modmin)
01342
01343
                    cx++;
01344
                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01345
                            modmean[ix][iy][iz] < ctl->csi_modmin)
01346
                   cy++;
01347
                 01348
01349
                    cz++;
               }
01351
01352
01353
        /* Write output... */
        if (fmod(t, ctl->csi_dt_out) == 0) {
01354
01355
           01356
01357
                   (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,

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

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

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

(cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01358
01359
01360
01361
01362
01363
01364
           /* Set counters to zero... */
01365
          cx = cy = cz = 0;
01366
01367
```

```
/* Close file... */
01369
        if (t == ctl->t_stop)
01370
           fclose(out);
01371 }
01372
01374
01375 void write_grid(
01376 const char *filename,
01377
         ctl_t * ctl,
        met_t * met0,
01378
01379
        met_t * met1,
        atm_t * atm,
01380
01381
        double t) {
01382
01383
        FILE *in, *out;
01384
01385
        char line[LEN];
01386
01387
        static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01388
         area, rho_air, press, temp, cd, mmr, t0, t1, r;
01389
01390
        static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01391
01392
         /* Check dimensions... */
        if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
01393
01394
           ERRMSG("Grid dimensions too large!");
01395
01396
        /* Check quantity index for mass... */
01397
        if (ctl->gnt m < 0)
01398
          ERRMSG("Need quantity mass to write grid data!");
01399
01400
         /\star Set time interval for output... \star/
01401
         t0 = t - 0.5 * ctl->dt_mod;
         t1 = t + 0.5 * ctl->dt_mod;
01402
01403
01404
         /* Set grid box size... */
        dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
01405
        dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
01406
01407
01408
01409
         /* Initialize grid... */
        for (ix = 0; ix < ctl->grid_nx; ix++)
01410
           for (iy = 0; iy < ctl->grid_ny; iy++)
01411
             for (iz = 0; iz < ctl->grid_nz; iz++)
01412
01413
                grid_m[ix][iy][iz] = 0;
01414
        /* Average data... */
for (ip = 0; ip < atm->np; ip++)
  if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {</pre>
01415
01416
01417
01418
01419
             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);
01420
01421
01422
01423
             /* Check indices... */
01425
             if (ix < 0 || ix >= ctl->grid_nx ||
01426
                 iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
               continue;
01427
01428
01429
             /* Add mass... */
01430
             grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01431
01432
        /* Check if gnuplot output is requested... */ if (ctl->grid_gpfile[0] != '-') {
01433
01434
01435
01436
           /* Write info... */
          printf("Plot grid data: %s.png\n", filename);
01437
01438
01439
           /* Create gnuplot pipe... */
           if (!(out = popen("gnuplot", "w")))
01440
             ERRMSG("Cannot create pipe to gnuplot!");
01441
01442
           /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
01443
01444
01445
01446
           /* Set time string... */
           jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01447
01448
01449
                    year, mon, day, hour, min);
01450
01451
           /* Dump gnuplot file to pipe... */
           if (!(in = fopen(ctl->grid_gpfile, "r")))
    ERRMSG("Cannot open file!");
01452
01453
01454
           while (fgets(line, LEN, in))
```

```
fprintf(out, "%s", line);
          fclose(in);
01456
01457
01458
01459
        else (
01460
           /* Write info... */
01461
01462
          printf("Write grid data: %s\n", filename);
01463
01464
           /* Create file... */
          if (!(out = fopen(filename, "w")))
01465
            ERRMSG("Cannot create file!");
01466
01467
01468
01469
        /* Write header... */
01470
        fprintf(out,
                  "# $1 = time [s] \n"
01471
                 "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n"
01472
01473
01474
                  "# $4 = latitude [deg] n"
01475
                  "# $5 = surface area [km^2]\n"
                  "# $6 = layer width [km] \n"
01476
                  "# $7 = temperature [K]\n"
01477
                  "# $8 = \text{column density } [kg/m^2] \n"
01478
01479
                  "# $9 = mass mixing ratio [1]\n\n");
01480
01481
         /* Write data... */
01482
        for (ix = 0; ix < ctl->grid_nx; ix++) {
01483
          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)
01484
01485
01486
01487
               fprintf(out, "\n");
01488
             for (iz = 0; iz < ctl->grid_nz; iz++)
01489
               if (!ctl->grid_sparse
                    || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01490
01491
                 /* Set coordinates... */
01493
                  z = ctl->grid_z0 + dz * (iz + 0.5);
                 lon = ctl->grid_lon0 + dlon * (ix + 0.5);
lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01494
01495
01496
                 /* Get pressure and temperature... */
press = P(z);
01497
01498
01499
                  intpol_met_time(met0, met1, t, press, lon, lat,
01500
                                   NULL, &temp, NULL, NULL, NULL, NULL, NULL);
01501
01502
                 /* Calculate surface area... */
                 area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)

* cos(lat * M_PI / 180.);
01503
01504
01506
                  /* Calculate column density... */
01507
                  cd = grid_m[ix][iy][iz] / (le6 * area);
01508
                 /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01509
01510
01511
01512
                 01513
01514
01515
01516
               }
01517
          }
01518
01519
01520
        /* Close file... */
01521
        fclose(out);
01522 }
01523
01525
01526 void write_prof(
01527
        const char *filename,
01528
        ctl_t * ctl,
met_t * met0,
01529
01530
        met_t * met1,
01531
01532
        double t) {
01533
01534
        static FILE *in. *out:
01535
01536
        static char line[LEN];
01537
01538
        static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
01539
          rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z,
01540
           press, temp, rho_air, mmr, h2o, o3;
01541
```

```
static int init, obscount[GX][GY], ip, ix, iy, iz;
01543
01544
         /* Init... */
         if (!init) {
01545
01546
           init = 1;
01547
01548
            /\star Check quantity index for mass... \star/
01549
            if (ctl->qnt_m < 0)
01550
             ERRMSG("Need quantity mass!");
01551
01552
            /* Check dimensions... */
           if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
01553
01554
             ERRMSG("Grid dimensions too large!");
01555
01556
            /\star Open observation data file... \star/
           printf("Read profile observation data: %s\n", ctl->prof_obsfile);
if (!(in = fopen(ctl->prof_obsfile, "r")))
01557
01558
             ERRMSG("Cannot open file!");
01559
01560
01561
            /* Create new file... */
           printf("Write profile data: %s\n", filename);
if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");
01562
01563
01564
01565
01566
            /* Write header... */
01567
           fprintf(out,
                             = time [s]\n"
01568
                     "# $1
                     "# $2 = altitude [km]\n"
01569
                     "# $3 = longitude [deg]\n"
"# $4 = latitude [deg]\n"
"# $5 = pressure [hPa]\n"
01570
01571
01572
01573
                     "# $6
                            = temperature [K]\n"
01574
                     "# $7 = mass mixing ratio [1]\n"
01575
                     "# $8 = H20 volume mixing ratio [1]\n"
                     "# $9 = O3 volume mixing ratio [1]\n"
01576
                     "# $10 = mean BT index [K]\n");
01577
01578
01579
            /* Set grid box size... */
01580
           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;
01581
01582
01583
01584
01585
         /* Set time interval... */
         t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01586
01587
01588
01589
         /* Initialize... */
         for (ix = 0; ix < ctl->prof_nx; ix++)
01590
01591
          for (iy = 0; iy < ctl->prof_ny; iy++) {
              obsmean[ix][iy] = 0;
01592
01593
              obscount[ix][iy] = 0;
              tmean[ix][iy] = 0;
for (iz = 0; iz < ctl->prof_nz; iz++)
01594
01595
01596
                mass[ix][iy][iz] = 0;
01597
           }
01598
01599
         /* Read data... */
01600
         while (fgets(line, LEN, in)) {
01601
01602
            /* Read data... */
           if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01603
01604
             continue;
01605
01606
            /* Check time... */
01607
           <u>if</u> (rt < t0)
01608
           continue;
if (rt > t1)
01609
01610
             break:
01611
01612
            /* Calculate indices... */
           ix = (int) ((rlon - ctl->prof_lon0) / dlon);
iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01613
01614
01615
           /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01616
01617
01618
              continue;
01619
01620
            /* Get mean observation index... */
           obsmean[ix][iy] += robs;
tmean[ix][iy] += rt;
01621
01622
01623
           obscount[ix][iy]++;
01624
01625
01626
         /\star Analyze model data... \star/
01627
         for (ip = 0; ip < atm->np; ip++) {
01628
```

```
/* Check time... */
01630
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
01631
             continue;
01632
           /* Get indices... */
ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
01633
01634
01635
01636
           iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01637
           /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx ||
    iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01638
01639
01640
01641
             continue;
01642
01643
           /\star Get total mass in grid cell... \star/
01644
          mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01645
01646
01647
        /* Extract profiles... */
        for (ix = 0; ix < ctl->prof_nx; ix++)
01649
          for (iy = 0; iy < ctl->prof_ny; iy++)
01650
             if (obscount[ix][iy] > 0) {
01651
                /* Write output... */
01652
               fprintf(out, "\n");
01653
01654
01655
                / \star \ \texttt{Loop over altitudes...} \ \star /
01656
                for (iz = 0; iz < ctl->prof_nz; iz++) {
01657
01658
                  /* Set coordinates... */
01659
                  z = ctl - prof_z0 + dz * (iz + 0.5);
01660
                  lon = ctl - > prof_lon0 + dlon * (ix + 0.5);
01661
                  lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01662
01663
                  /* Get meteorological data... */
                  press = P(z);
01664
                  intpol_met_time(met0, met1, t, press, lon, lat,
01665
                                    NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
01666
01667
                 /* 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.);
01668
01669
01670
01671
01672
                 mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
01673
01674
                  /* Write output... */
                  fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n", tmean[ix][iy] / obscount[ix][iy],
01675
01676
                           z, lon, lat, press, temp, mmr, h2o, o3, obsmean[ix][iy] / obscount[ix][iy]);
01677
01678
01679
               }
01680
01681
01682
        /* Close file... */
        if (t == ctl->t_stop)
01683
          fclose(out);
01684
01685 }
01686
01688
01689 void write station(
01690 const char *filename,
        ctl_t * ctl,
atm_t * atm,
01691
01692
01693
        double t) {
01694
01695
        static FILE *out:
01696
01697
        static double rmax2, t0, t1, x0[3], x1[3];
01699
        static int init, ip, iq;
01700
        /* Init... */
if (!init) {
01701
01702
01703
          init = 1;
01704
01705
           /* Write info... */
01706
          printf("Write station data: %s\n", filename);
01707
01708
           /* Create new file... */
01709
           if (!(out = fopen(filename, "w")))
             ERRMSG("Cannot create file!");
01711
01712
           /* Write header... */
           fprintf(out,
01713
                    "# $1 = time [s]\n"
01714
01715
                    "# $2 = altitude [km] \n"
```

```
01716
                   "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
          01717
01718
01719
01720
01721
01722
           /\star Set geolocation and search radius... \star/
01723
          geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
01724
          rmax2 = gsl_pow_2(ctl->stat_r);
01725
01726
        /* Set time interval for output... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01727
01728
01729
01730
        /* Loop over air parcels... */
for (ip = 0; ip < atm->np; ip++) {
01731
01732
01733
01734
          /* Check time... */
01735
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
01736
01737
01738
          /* Check station flag... */
01739
          if (ctl->qnt_stat >= 0)
  if (atm->q[ctl->qnt_stat][ip])
01740
01741
              continue;
01742
01743
           /\star Get Cartesian coordinates... \star/
01744
          geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
01745
01746
          /* Check horizontal distance... */
01747
          if (DIST2(x0, x1) > rmax2)
01748
            continue;
01749
          /* Set station flag... */
if (ctl->qnt_stat >= 0)
atm->q[ctl->qnt_stat][ip] = 1;
01750
01751
01752
01753
          01754
01755
01756
          for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01757
01758
01759
01760
          fprintf(out, "\n");
01761
01762 }
01763
        /* Close file... */
01764
        if (t == ctl->t_stop)
01765
01766
          fclose(out);
01767 }
```

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 (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, int np, float dest[EX][EY][EP], float scl)

Read and convert variable from meteorological data file.

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_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.

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

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.

```
00085 {
00086
00087 return dy * 180. / (M_PI * RE);
00088 }
```

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.

```
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);
```

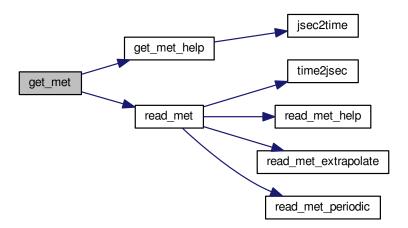
```
5.13.2.9 void get_met ( ctl_t * ctl, char * metbase, double t, met_t * met0, met_t * met1 )
```

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
00128
       /* Init... */
00129
       if (!init) {
00130
         init = 1;
00131
         get_met_help(t, -1, metbase, ctl->dt_met, filename);
00132
00133
         read_met(filename, met0);
00134
00135
          get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
     dt_met, filename);
00136
         read_met(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));
00142
          get_met_help(t, 1, metbase, ctl->dt_met, filename);
         read_met(filename, met1);
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(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.

```
00161
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)
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
        /* Set filename... */
sprintf(filename, "%s_%d_%02d_%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.

```
{
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 }
```

5.13.2.12 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
                         {
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 }
```

5.13.2.13 void intpol_met_space ($met_t * met$, double p, double lon, double lo

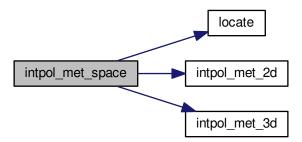
Spatial interpolation of meteorological data.

Definition at line 236 of file libtrac.c.

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



5.13.2.14 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 * b2o, double * o3)

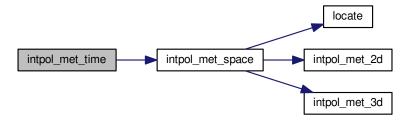
Temporal interpolation of meteorological data.

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:



5.13.2.15 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
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 }
```

```
5.13.2.16 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;
        ihi = n - 1;
i = (ihi + ilo) >> 1;
00384
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
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 }
```

5.13.2.17 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
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 }
```

5.13.2.18 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
00465
00466
        /* Write info... */
        printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
00467
                 "(executable: %s | compiled: %s, %s)\n\n",
00468
                argv[0], __DATE__, __TIME__);
00470
00471
        /* Initialize quantity indices... */
        ctl->qnt_m = -1;
ctl->qnt_r = -1;
00472
00473
00474
        ctl->qnt_rho = -1;
        ct1->qnt_ps = -1;
00475
        ctl->qnt_t = -1;
00476
        ctl->qnt_u = -1;
00477
        ctl->qnt_v = -1;
00478
        ctl \rightarrow qnt_w = -1;
00479
00480
        ct1->ant h2o = -1;
00481
        ctl->qnt_o3 = -1;
00482
        ctl->qnt_theta = -1;
00483
        ctl->qnt\_stat = -1;
00484
00485
        /* Read quantities... */
        ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
00486
        for (iq = 0; iq < ctl->nq; iq++) {
00487
          /* 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",
00489
00490
00491
00492
                    ctl->gnt format[ig]);
00493
           /\star Try to identify quantity... \star/
00494
00495
           if (strcmp(ctl->qnt_name[iq], "m") == 0) {
00496
            ctl->qnt_m = iq;
00497
             sprintf(ctl->qnt_unit[iq], "kg");
          } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
  ctl->qnt_r = iq;
00498
00499
            sprintf(ctl->qnt_unit[iq], "m");
00501
           } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
00502
             ctl->qnt_rho = iq;
00503
            sprintf(ctl->qnt_unit[iq], "kg/m^3");
00504
           } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
            ctl->qnt_ps = iq;
00505
             sprintf(ctl->qnt_unit[iq], "hPa");
00506
          } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
00507
           ctl->qnt_t = iq;
00508
00509
             sprintf(ctl->qnt_unit[iq], "K");
          } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
ctl->qnt_u = iq;
00510
00511
             sprintf(ctl->qnt_unit[iq], "m/s");
00513
          } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
00514
            ctl->qnt_v = iq;
00515
            sprintf(ctl->qnt_unit[iq], "m/s");
          } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
  ctl->qnt_w = iq;
  sprintf(ctl->qnt_unit[iq], "hPa/s");
00516
00517
00518
          } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
00520
           ctl->qnt_h2o = iq;
00521
             sprintf(ctl->qnt_unit[iq], "1");
00522
          } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
00523
            ctl->qnt_o3 = iq;
            sprintf(ctl->qnt_unit[iq], "1");
00524
          } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
  ctl->qnt_theta = iq;
  sprintf(ctl->qnt_unit[iq], "K");
00525
00526
00527
           } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
00528
             ctl->qnt_stat = iq;
00529
             sprintf(ctl->qnt_unit[iq], "-");
00530
00531
00532
             scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00533
00534
        /\star Time steps of simulation... \star/
00535
00536
        ctl->direction =
00537
          (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
        if (ctl->direction != -1 && ctl->direction != 1)
```

```
00539
           ERRMSG("Set DIRECTION to -1 or 1!");
00540
         ctl->t_start =
           scan_ctl(filename, argc, argv, "T_START", -1, "-1e100", NULL);
00541
         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);
00542
00543
00544
00545
          /* Meteorological data... */
00546
         ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
00547
00548
         /* Isosurface parameters... */
00549
         ctl->isosurf
         = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00550
00551
00552
00553
         /* Diffusion parameters... */
00554
         ctl->turb_dx_trop
           = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00555
00556
         ctl->turb dx strat
           = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00558
         ctl->turb dz trop
00559
            = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00560
         ctl->turb_dz_strat
           = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00561
00562
         ct.1->t.urb meso =
00563
           scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00564
00565
         /* Life time of particles... */
00566
         ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
         ctl->tdec strat =
00567
           scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00568
00569
        /* Output of atmospheric data... */
scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
00570
00571
       atm_basename);
00572
         scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00573
         ctl->atm dt out
00574
           scan ctl(filename, argc, argv, "ATM DT OUT", -1, "86400", NULL);
00575
00576
         /* Output of CSI data... */
         scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
00577
      csi_basename);
scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "obs.tab",
00579
00580
                   ctl->csi_obsfile);
00581
00582
         ctl->csi_obsmin =
00583
           scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00584
         ctl->csi modmin =
           scan ctl(filename, argc, argv, "CSI MODMIN", -1, "0", NULL);
00585
         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);
00586
         ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
00588
00589
         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);
00590
00591
00592
         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);
00593
00594
00595
00596
         ctl->csi_ny =
00597
            (int) scan ctl(filename, argc, argv, "CSI NY", -1, "180", NULL);
00598
00599
         /* Output of grid data... */
00600
         scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
00601
                    ctl->grid_basename);
00602
         scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
grid_gpfile);
00603 ctl=>-
         ctl->grid_dt_out =
00604
           scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
00605
         ctl->grid_sparse =
00606
           (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
         ctl-ygrid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
ctl-ygrid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
00607
00608
00609
         ctl->grid_nz =
            (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00610
00611
         ctl->grid lon0 =
00612
           scan_ctl(filename, argc, argv, "GRID_LONO", -1, "-180", NULL);
00613
         ctl->grid_lon1
           scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
00614
00615
         ctl->grid nx =
           (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00616
00617
         ctl->grid_lat0 =
00618
            scan_ctl(filename, argc, argv, "GRID_LATO", -1, "-90", NULL);
00619
         ctl->grid_lat1
00620
           scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
         ctl->grid_ny =
00621
00622
           (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
```

```
00623
         /* Output of profile data... */
00624
         scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
00625
00626
                   ctl->prof_basename);
         scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
00627
prof_obsfile);

00628 ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);

00629 ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
00630
         ctl->prof_nz =
00631
           (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
00632
         ctl->prof_lon0 =
           scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
00633
00634
        ctl->prof lon1
00635
           scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
00636
00637
           (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00638
        ctl->prof_lat0 =
           scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
00639
00640
         ctl->prof_lat1 =
00641
           scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
00642
         ctl->prof_ny =
00643
           (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00644
        /* Output of station data... */
00645
        00646
        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);
00648
00649
00650
00651 }
```

Here is the call graph for this function:



```
5.13.2.19 void read_met ( char * filename, met_t * met )
```

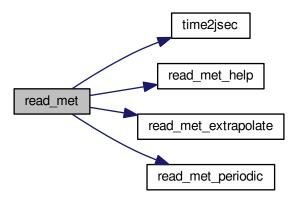
Read meteorological data file.

Definition at line 655 of file libtrac.c.

```
00657
00658
00659
       char tstr[10]:
00660
00661
        static float help[EX * EY];
00662
00663
       int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00664
00665
       size t np, nx, nv;
00666
00667
00668
       printf("Read meteorological data: %s\n", filename);
00669
00670
        /* Get time from filename... */
        sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00671
00672
        year = atoi(tstr);
00673
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00674
        mon = atoi(tstr);
00675
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00676
        day = atoi(tstr);
sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00677
00678
        hour = atoi(tstr);
        time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
```

```
/* Open netCDF file... */
NC(nc_open(filename, NC_NOWRITE, &ncid));
00681
00682
00683
          /* Get dimensions... */
NC(nc_inq_dimid(ncid, "lon", &dimid));
00684
00685
          NC(nc_inq_dimlen(ncid, dimid, &nx));
00687
              (nx > EX)
00688
            ERRMSG("Too many longitudes!");
00689
          NC(nc_inq_dimid(ncid, "lat", &dimid));
NC(nc_inq_dimlen(ncid, dimid, &ny));
00690
00691
00692
          if (ny > EY)
00693
             ERRMSG("Too many latitudes!");
00694
00695
          NC(nc_inq_dimid(ncid, "lev", &dimid));
00696
          NC(nc_inq_dimlen(ncid, dimid, &np));
00697
          if (np > EP)
            ERRMSG("Too many pressure levels!");
00698
00699
00700
          /* Store dimensions... */
          met->np = (int) np;
met->nx = (int) nx;
00701
00702
00703
          met->ny = (int) ny;
00704
00705
          /* Read geolocations... */
NC(nc_inq_varid(ncid, "lev", &varid));
00706
00707
          NC(nc_get_var_double(ncid, varid, met->p));
00708
          NC(nc_inq_varid(ncid, "lon", &varid));
00709
00710
          NC(nc_get_var_double(ncid, varid, met->lon));
00711
00712
          NC(nc_inq_varid(ncid, "lat", &varid));
00713
          NC(nc_get_var_double(ncid, varid, met->lat));
00714
          /\star Check and convert pressure levels... \star/
00715
00716
          for (ip = 0; ip < met->np; ip++) {
   if (ip > 0 && met->p[ip - 1] > met->p[ip])
00718
               ERRMSG("Pressure levels must be descending!");
00719
            met->p[ip] /= 100.;
00720
00721
00722
          /* Read surface pressure... */
          if (nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00723
00724
            NC(nc_get_var_float(ncid, varid, help));
             for (iy = 0; iy < met->ny; iy++)
for (ix = 0; ix < met->nx; ix++)
00725
00726
                 met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00727
00728
          } else {
00729
            for (ix = 0; ix < met->nx; ix++)
               for (iy = 0; iy < met->ny; iy++)
00730
00731
                 met->ps[ix][iy] = met->p[0];
00732
00733
          /* Read meteorological data... */
read_met_help(ncid, "t", "T", met, met->np, met->t, 1.0);
read_met_help(ncid, "u", "U", met, met->np, met->u, 1.0);
00734
00735
         read_met_help(ncid, "v", "V", met, met->np, met->v, 1.0);
read_met_help(ncid, "w", "V", met, met->np, met->v, 1.0);
read_met_help(ncid, "w", "W", met, met->np, met->v, 0.01f);
read_met_help(ncid, "q", "Q", met, met->np, met->holo, 1.608f);
read_met_help(ncid, "o3", "03", met, met->np, met->o3, 0.602f);
00737
00738
00739
00740
00741
00742
          /* Extrapolate data for lower boundary... */
00743
          read_met_extrapolate(met);
00744
00745
          /\star Copy data to obtain periodic boundary conditions... \star/
00746
         read_met_periodic(met);
00747
00748
          /* Close file... */
00749
          NC(nc_close(ncid));
00750 }
```

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 754 of file libtrac.c.

```
00755
                         {
00756
00757
         int ip, ip0, ix, iy;
00758
00759
         /* Loop over columns... */
00760
        for (ix = 0; ix < met->nx; ix++)
00761
           for (iy = 0; iy < met->ny; iy++) {
00762
00763
              /* Find lowest valid data point... */
             for (ip0 = met->np - 1; ip0 >= 0; ip0--)
   if (!gsl_finite(met->t[ix][iy][ip0])
00764
00765
00766
                     || !gsl_finite(met->u[ix][iy][ip0])
00767
                     || !gsl_finite(met->v[ix][iy][ip0])
00768
                     || !gsl_finite(met->w[ix][iy][ip0]))
00769
                  break:
00770
             /* Extrapolate... */
for (ip = ip0; ip >= 0; ip--) {
00771
00772
00773
               met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
00774
               met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
00775
               met \rightarrow v[ix][iy][ip] = met \rightarrow v[ix][iy][ip + 1];
               met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
00776
00777
00778
                met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00779
00780
            }
00781 }
```

5.13.2.21 void read_met_help (int *ncid*, char * *varname*, char * *varname*2, met_t * *met*, int *np*, float *dest[EX][EY][EP]*, float *scl*)

Read and convert variable from meteorological data file.

Definition at line 785 of file libtrac.c.

```
00792
                     {
00793
00794
         static float help[EX * EY * EP];
00795
00796
         int ip, ix, iy, n = 0, varid;
00797
00798
         /* Check if variable exists... */
00799
         if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00800
          if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00801
00802
00803
         /* Read data... */
         NC(nc_get_var_float(ncid, varid, help));
00804
00805
00806
         /* Copy and check data...
00807
         for (ip = 0; ip < np; ip++)</pre>
           for (iy = 0; iy < met->ny; iy++)
for (ix = 0; ix < met->nx; ix++) {
00808
00809
               dest[ix][iy][ip] = scl * help[n++];
if (dest[ix][iy][ip] < -le10 || dest[ix][iy][ip] > le10)
00810
00811
00812
                  dest[ix][iy][ip] = GSL_NAN;
00813
00814 }
```

5.13.2.22 void read_met_periodic (met_t * met)

Create meteorological data with periodic boundary conditions.

Definition at line 818 of file libtrac.c.

```
00819
                                {
00820
00821
           int ip, iy;
00822
00823
            /* Check longitudes... */
           if (fabs(met->lon[met->nx - 1] - met->lon[0] - 360) < 0.01)
00824
00825
              return;
00826
00827
           /* Increase longitude counter... */
00828
           if ((++met->nx) > EX)
00829
              ERRMSG("Cannot create periodic boundary conditions!");
00830
          /* Set longitude... */
met->lon[met->nx - 2] + met->lon[1] - met->
00831
00832
        lon[0];
00833
00834
            /* 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];
00835
00836
00837
                 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->v[0][iy][ip];
met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
00838
00839
00840
00841
                 met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
00842
00843
00845 }
```

5.13.2.23 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 849 of file libtrac.c.

```
00856 {
00857
00858 FILE *in = NULL;
00859
00860 char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
00861 msg[LEN], rvarname[LEN], rval[LEN];
```

```
int contain = 0, i;
00865
          /* Open file... */
         if (filename[strlen(filename) - 1] != '-')
  if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");
00866
00867
00868
00870
         /* Set full variable name... */
00871
         if (arridx >= 0) {
           sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
00872
00873
00874
         } else {
         sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
00875
00876
00877
00878
00879
         /* Read data... */
00880
         if (in != NULL)
00881
           while (fgets(line, LEN, in))
             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
if (strcasecmp(rvarname, fullname1) == 0 ||
00883
00884
                      strcasecmp(rvarname, fullname2) == 0) {
00885
                   contain = 1;
00886
                   break;
00887
                 }
         for (i = 1; i < argc - 1; i++)</pre>
          if (strcasecmp(argv[i], fullname1) == 0 ||
00889
              strcasecmp(argv[i], fullname2) == 0) {
sprintf(rval, "%s", argv[i + 1]);
00890
00891
00892
              contain = 1;
00893
              break:
00894
           }
00895
00896
         /* Close file... */
00897
         if (in != NULL)
           fclose(in);
00898
00899
00900
         /* Check for missing variables... */
00901
         if (!contain) {
         if (strlen(defvalue) > 0)
    sprintf(rval, "%s", defvalue);
00902
00903
           else {
00904
             sprintf(msg, "Missing variable %s!\n", fullname1);
00905
00906
              ERRMSG (msg);
00907
00908
00909
        /* Write info... */
printf("%s = %s\n", fullname1, rval);
00910
00911
00912
         /* Return values... */
00914
         if (value != NULL)
00915
           sprintf(value, "%s", rval);
00916
         return atof(rval);
00917 }
```

5.13.2.24 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 921 of file libtrac.c.

```
00929
00931
        struct tm t0, t1;
00932
00933
        t0.tm_year = 100;
        t0.tm\_mon = 0;
00934
        t0.tm_mday = 1;
t0.tm_hour = 0;
00935
00936
00937
        t0.tm_min = 0;
00938
        t0.tm\_sec = 0;
00939
00940
        t1.tm year = year - 1900;
        t1.tm_mon = mon - 1;
00941
00942
        t1.tm_mday = day;
00943
        t1.tm_hour = hour;
        t1.tm_min = min;
00944
        t1.tm_sec = sec;
00945
00946
00947
        *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
00948 }
```

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

Measure wall-clock time.

Definition at line 952 of file libtrac.c.

```
00955
00956
00957
       static double starttime[NTIMER], runtime[NTIMER];
00958
        /* Check id...
00959
00960
        if (id < 0 || id >= NTIMER)
         ERRMSG("Too many timers!");
00961
00962
00963
        /* Start timer... */
00964
       if (mode == 1) {
        if (starttime[id] <= 0)</pre>
00965
00966
            starttime[id] = omp_get_wtime();
00967
           ERRMSG("Timer already started!");
00968
00969
00970
00971
        /* Stop timer... */
00972
        else if (mode == 2) {
00973
         if (starttime[id] > 0) {
00974
           runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
            starttime[id] = -1;
00975
00976
         } else
00977
            ERRMSG("Timer not started!");
00978
00979
        /* Print timer... ∗/
00980
       else if (mode == 3)
00981
         printf("%s = %g s\n", name, runtime[id]);
00982
```

5.13.2.26 double tropopause (double t, double lat)

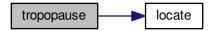
Definition at line 987 of file libtrac.c.

```
00989
00990
00991
            static double doys[12]
             = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00992
00993
00994
             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,
00995
00996
               -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,
00997
00998
01000
01001
                75, 77.5, 80, 82.5, 85, 87.5, 90
01002
            };
01003
            static double tps[12][73]
01004
             = { {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,
01005
                         175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
01007
01008
                         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,
01009
01010
01011
             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,
01012
01013
01014
              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, 98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27, 98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193, 220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
01015
01016
01017
01019
              284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01020
               287.5, 286.2, 285.8},
01021
             {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, 100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01022
01023
01024
              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,
01027
01028
            304.3, 304.9, 306, 306.6, 306.2, 306},
01029
          {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,
01030
01031
           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,
01033
01034
           148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
01035
           263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4
          315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1, 
{266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8, 
260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
01036
01037
                    258.6,
           205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
01039
01040
           101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
           102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9, 165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
01041
01042
           273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01043
           325.3, 325.8, 325.8},
          {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
01045
           222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
01046
01047
           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, 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,
01048
01049
01050
           308.5, 312.2, 313.1, 313.3},
01052
01053
          {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
           187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6, 235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1, 110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7, 111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
01054
01055
01056
           117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
01058
01059
           224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5
01060
           275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
          {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01061
           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,
01062
01064
            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
01065
01066
           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,
01067
          278.2, 282.6, 287.4, 290.9, 292.5, 293}, {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
01068
01069
           183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
01070
01071
           243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
01072
           114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
           110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6, 114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9, 203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
01073
01074
01075
            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,
01077
01078
           215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
           237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2, 111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01079
01080
           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,
01081
           206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01083
           279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
01084
           305.1},
01085
          {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01086
           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,
01087
           108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5,
           102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
01090
           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,
01091
01092
           286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6}
01093
          {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
01094
           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,
01096
01097
           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,
01098
01099
01100
           280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
           281.7, 281.1, 281.2}
01102
01103
01104
          double doy, p0, p1, pt;
01105
01106
          int imon, ilat;
          /* Get day of year... */
doy = fmod(t / 86400., 365.25);
01108
01109
01110
          while (doy < 0)
01111
            dov += 365.25;
01112
```

```
01113
         /* Get indices... */
         imon = locate(doys, 12, doy);
ilat = locate(lats, 73, lat);
01114
01115
01116
         /* Get tropopause pressure... */
01117
         p0 = LIN(lats[ilat], tps[imon][ilat],
lats[ilat + 1], tps[imon][ilat + 1], lat);
01118
01119
         p1 = LIN(lats[ilat], tps[imon + 1][ilat],
lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
01120
01121
01122
         pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01123
01124
         /* Return tropopause pressure... */
01125
         return pt;
01126 }
```

Here is the call graph for this function:



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

Write atmospheric data.

Definition at line 1130 of file libtrac.c.

```
01134
01135
01136
        FILE *in, *out;
01137
        char line[LEN];
01138
01139
01140
        double r;
01141
01142
         int ip, iq, year, mon, day, hour, min, sec;
01143
         /* Check if gnuplot output is requested... */ if (ctl->atm_gpfile[0] != '-') {
01144
01145
01146
01147
           /\star Write info... \star/
01148
           printf("Plot atmospheric data: %s.png\n", filename);
01149
          /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
01150
01151
             ERRMSG("Cannot create pipe to gnuplot!");
01152
01153
           01154
01155
01156
01157
           /* Set time string... */
           jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01158
01159
01160
                    year, mon, day, hour, min);
01161
01162
           /* Dump gnuplot file to pipe... */
           if (!(in = fopen(ctl->atm_gpfile, "r")))
ERRMSG("Cannot open file!");
01163
01164
           while (fgets(line, LEN, in))
fprintf(out, "%s", line);
01165
01166
01167
           fclose(in);
01168
        }
01169
01170
        else {
01171
01172
          /* Write info... */
```

```
printf("Write atmospheric data: %s\n", filename);
01174
01175
          /* Create file... */
         if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");
01176
01177
01178
01179
01180
        /* Write header... */
01181
       fprintf(out,
                "# $1 = time [s]\n"
"# $2 = altitude [km]\n"
01182
01183
                "# \$3 = longitude [deg] \n" "# \$4 = latitude [deg] \n");
01184
       01185
01186
01187
       fprintf(out, "\n");
01188
01189
        /* Write data... */
01190
       01191
01192
01193
         for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01194
01195
01196
01197
01198
         fprintf(out, "\n");
01199
01200
01201
       /* Close file... */
01202
       fclose(out);
01203 }
```

Here is the call graph for this function:



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

Write CSI data.

Definition at line 1207 of file libtrac.c.

```
01211
01212
       static FILE *in, *out;
01214
01215
       static char line[LEN];
01216
       static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01217
01218
         rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01219
01220
       static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01221
        /* Init... */
01222
01223
        if (!init) {
01224
         init = 1;
01225
01226
          /\star Check quantity index for mass... \star/
01227
         if (ctl->qnt_m < 0)
           ERRMSG("Need quantity mass to analyze CSI!");
01228
01229
01230
          /* Open observation data file... */
01231
         printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
01232
          if (!(in = fopen(ctl->csi_obsfile, "r")))
```

```
01233
           ERRMSG("Cannot open file!");
01234
01235
          /* Create new file... */
         01236
01237
           ERRMSG("Cannot create file!");
01238
01239
01240
          /* Write header... */
01241
          fprintf(out,
01242
                  "# $1 = time [s] \n"
                  "# $2 = number of hits (cx)\n"
01243
                  "# $3 = number of misses (cy)\n"
01244
01245
                  "# $4 = number of false alarms (cz)\n"
01246
                  "# $5 = number of observations (cx + cy) n"
01247
                  "# $6 = number of forecasts (cx + cz)\n"
01248
                  "# \$7 = bias (forecasts/observations) [\%] \n"
                  "# $8 = probability of detection (POD) [%%]\n" "# $9 = false alarm rate (FAR) [%%]\n" "# $10 = critical success index (CSI) [%%]\n\n");
01249
01250
01251
01252
       }
01253
01254
       /* Set time interval... */
       t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01255
01256
01257
01258
        /* Initialize grid cells... */
01259
        for (ix = 0; ix < ctl->csi_nx; ix++)
01260
         for (iy = 0; iy < ctl->csi_ny; iy++)
01261
            for (iz = 0; iz < ctl->csi_nz; iz++)
01262
             modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01263
01264
        /* Read data... */
01265
        while (fgets(line, LEN, in)) {
01266
          /* Read data... *,
01267
          if (sscanf(line, "%lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01268
01269
              5)
01270
            continue;
01271
01272
          /* Check time... */
01273
         <u>if</u> (rt < t0)
01274
           continue;
          if (rt > t1)
01275
01276
           break;
01277
01278
          /* Calculate indices... */
01279
          ix = (int) ((rlon - ctl->csi_lon0))
                      / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01280
          iy = (int) ((rlat - ctl -> csi_lat0))
01281
01282
                      / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01283
         iz = (int) ((rz - ctl -> csi_z0)
01284
                      / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01285
         /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
01286
01287
             iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01288
            continue:
01290
01291
          /* Get mean observation index... */
01292
         obsmean[ix][iy][iz] += robs;
01293
         obscount[ix][iy][iz]++;
01294
01295
01296
        /* Analyze model data... */
01297
        for (ip = 0; ip < atm->np; ip++) {
01298
         01299
01300
01301
           continue;
01302
01303
          /* Get indices... */
01304
          ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
                      / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01305
         01306
01307
         01308
01309
01310
         /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
    iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01311
01312
01313
01314
            continue;
01315
01316
          /∗ Get total mass in grid cell... ∗/
01317
          modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01318
01319
```

```
/* Analyze all grid cells... */
         for (ix = 0; ix < ctl->csi_nx; ix++)
01321
01322
           for (iy = 0; iy < ctl->csi_ny; iy++)
             for (iz = 0; iz < ctl->csi_nz; iz++) {
01323
01324
01325
                /* Calculate mean observation index... */
                if (obscount[ix][iy][iz] > 0)
01326
                  obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01327
01328
01329
                /\star Calculate column density... \star/
                if (modmean[ix][iy][iz] > 0) {
01330
                 dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
01331
01332
01333
                  lat = ctl -> csi_lat0 + dlat * (iy + 0.5);
                 area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180. * cos(lat * M_PI / 180.);
01334
01335
                  modmean[ix][iy][iz] /= (1e6 * area);
01336
                }
01337
01338
01339
                /* Calculate CSI... */
01340
                if (obscount[ix][iy][iz] > 0) {
01341
                  if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01342
                      modmean[ix][iy][iz] >= ctl->csi_modmin)
01343
                    cx++;
01344
                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
                             modmean[ix][iy][iz] < ctl->csi_modmin)
01345
01346
                    cy++;
01347
                  else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
01348
                             modmean[ix][iy][iz] >= ctl->csi_modmin)
01349
                    cz++;
01350
01351
             }
01352
01353
         /* Write output... */
01354
        if (fmod(t, ctl->csi_dt_out) == 0) {
01355
           01356
01357
01358
                    t, cx, cy, cz, cx + cy, cx + cz,
                     t, cx, cy, cz, cx + cy, cx + cz,
(cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
(cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
(cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
(cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01359
01360
01361
01362
01363
01364
           /* Set counters to zero... */
01365
           cx = cy = cz = 0;
01366
01367
         /* Close file... */
01368
         if (t == ctl->t_stop)
01369
01370
           fclose(out);
01371 }
```

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

Write gridded data.

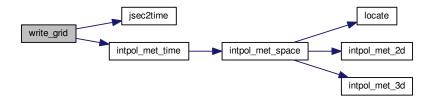
Definition at line 1375 of file libtrac.c.

```
01381
                  {
01382
01383
       FILE *in, *out;
01384
01385
       char line[LEN];
01386
01387
       static double grid m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01388
         area, rho air, press, temp, cd, mmr, t0, t1, r;
01389
01390
       static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01391
01392
        /* Check dimensions... */
        if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
01393
01394
         ERRMSG("Grid dimensions too large!");
01395
01396
        /* Check quantity index for mass... */
01397
       if (ctl->qnt_m < 0)
01398
         ERRMSG("Need quantity mass to write grid data!");
01399
01400
       /* Set time interval for output... */
01401
       t0 = t - 0.5 * ctl->dt_mod;
```

```
t1 = t + 0.5 * ctl->dt_mod;
01402
01403
01404
         /* 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;
01405
01406
01407
01408
01409
          /* Initialize grid... */
01410
         for (ix = 0; ix < ctl->grid_nx; ix++)
           for (iy = 0; iy < ctl->grid_ny; iy++)
  for (iz = 0; iz < ctl->grid_nz; iz++)
01411
01412
                grid_m[ix][iy][iz] = 0;
01413
01414
01415
         /* Average data... */
01416
         for (ip = 0; ip < atm->np; ip++)
           if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {</pre>
01417
01418
01419
              /* Get index... */
              ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
              iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
01421
01422
              iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01423
              /* Check indices... */
if (ix < 0 || ix >= ctl->grid_nx ||
    iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01424
01425
01426
01427
                continue:
01428
              /* Add mass... */
01429
              grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01430
01431
01432
         /* Check if gnuplot output is requested... */ if (ctl->grid_gpfile[0] != '-') {
01433
01434
01435
           /* Write info... */
printf("Plot grid data: %s.png\n", filename);
01436
01437
01438
01439
            /* Create gnuplot pipe... */
           if (!(out = popen("gnuplot", "w")))
01440
01441
             ERRMSG("Cannot create pipe to gnuplot!");
01442
           01443
01444
01445
01446
            /* Set time string... */
01447
            jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01448
           fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01449
                     year, mon, day, hour, min);
01450
01451
            /* Dump gnuplot file to pipe... */
           if (!(in = fopen(ctl->grid_gpfile, "r")))
    ERRMSG("Cannot open file!");
01452
01453
           while (fgets(line, LEN, in))
fprintf(out, "%s", line);
01454
01455
01456
           fclose(in);
01457
         }
01458
01459
01460
01461
            /* Write info... */
           printf("Write grid data: %s\n", filename);
01462
01463
01464
            /* Create file... */
01465
           if (!(out = fopen(filename, "w")))
01466
              ERRMSG("Cannot create file!");
01467
01468
         /* Write header... */
01469
01470
         fprintf(out,
                   "# $1 = time [s] \n"
01472
                   "# $2 = altitude [km] \n"
                   "# $3 = longitude [deg]\n"
01473
                   "# $4 = latitude [deg]\n"
01474
                   "# $5 = surface area [km^2]\n"
"# $6 = layer width [km]\n"
01475
01476
01477
                   "# $7 = temperature [K] \n"
                  "# $8 = column density [kg/m^2]\n"
"# $9 = mass mixing ratio [1]\n\n";
01478
01479
01480
01481
         /* Write data... */
         for (ix = 0; ix < ctl->grid_nx; ix++) {
01482
          if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01484
              fprintf(out, "\n");
01485
            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++)
01486
01487
01488
```

```
if (!ctl->grid_sparse
01490
                     || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01491
                   /* Set coordinates... */
01492
                   z = ctl->grid_z0 + dz * (iz + 0.5);
lon = ctl->grid_lon0 + dlon * (ix + 0.5);
01493
01494
01495
                   lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01496
01497
                   /\star Get pressure and temperature... \star/
01498
                   press = P(z);
                   01499
01500
01501
01502
                   /* Calculate surface area... */
                   area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
 * cos(lat * M_PI / 180.);
01503
01504
01505
                  /* Calculate column density... */
cd = grid_m[ix][iy][iz] / (le6 * area);
01506
01507
01508
                  /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01509
01510
01511
01512
01513
                    /* Write output... */
01514
                   fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g\n", t, z, lon, lat, area, dz, temp, cd, mmr);
01515
01516
01517
           }
01518
        }
01519
01520
          /* Close file... */
01521
       fclose(out);
01522 }
```

Here is the call graph for this function:



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

Write profile data.

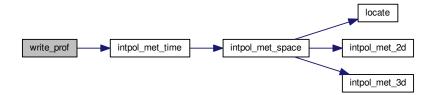
Definition at line 1526 of file libtrac.c.

```
01532
                   {
01533
01534
        static FILE *in, *out;
01535
        static char line[LEN];
01537
        static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
01538
         rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, mmr, h2o, o3;
01539
01540
01541
01542
        static int init, obscount[GX][GY], ip, ix, iy, iz;
01543
         /* Init... */
01544
        if (!init) {
01545
01546
          init = 1;
01547
01548
          /* Check quantity index for mass... */
```

```
if (ctl->qnt_m < 0)
01550
             ERRMSG("Need quantity mass!");
01551
           /* Check dimensions... */
if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
    ERRMSG("Grid dimensions too large!");
01552
01553
01554
01555
01556
            /* Open observation data file... */
01557
           printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01558
            if (!(in = fopen(ctl->prof_obsfile, "r")))
              ERRMSG("Cannot open file!");
01559
01560
           /* Create new file... */
01561
           printf("Write profile data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01562
01563
             ERRMSG("Cannot create file!");
01564
01565
01566
            /* Write header... */
01567
           fprintf(out,
                     "# $1
01568
                             = time [s]\n"
01569
                     "# $2
                            = altitude [km]\n"
                            = longitude [deg]\n"
01570
                     "# $3
                     "# $4 = latitude [deg]\n"
01571
                     "# $5
                            = pressure [hPa]\n"
01572
01573
                     "# $6 = temperature [K]\n"
01574
                     "# $7 = mass mixing ratio [1]\n"
01575
                     "# $8 = H20 volume mixing ratio [1]\n"
01576
                     "# $9 = 03 volume mixing ratio [1]\n"
                     "# $10 = mean BT index [K] \n");
01577
01578
01579
           /* Set grid box size... */
01580
           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;
01581
01582
01583
01584
         /* Set time interval... */
01585
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01587
01588
01589
         /* Initialize... */
         for (ix = 0; ix < ctl->prof_nx; ix++)
01590
          for (iy = 0; iy < ctl->prof_ny; iy++) {
01591
              obsmean[ix][iy] = 0;
01592
              obscount[ix][iy] = 0;
01593
01594
              tmean[ix][iy] = 0;
01595
              for (iz = 0; iz < ctl->prof_nz; iz++)
01596
                mass[ix][iy][iz] = 0;
01597
01598
01599
         /* Read data... */
01600
         while (fgets(line, LEN, in)) {
01601
           /* Read data... */
if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01602
01603
01604
             continue;
01605
01606
           /* Check time... */
01607
           if (rt < t0)</pre>
           continue;
if (rt > t1)
01608
01609
01610
             break;
01611
01612
            /* Calculate indices... */
           ix = (int) ((rlon - ctl->prof_lon0) / dlon);
iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01613
01614
01615
           /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01616
01617
             continue;
01619
01620
           /\star Get mean observation index... \star/
           obsmean[ix][iy] += robs;
tmean[ix][iy] += rt;
01621
01622
           obscount[ix][iy]++;
01623
01624
01625
01626
         /* Analyze model data... */
01627
         for (ip = 0; ip < atm->np; ip++) {
01628
           /* Check time... */
01629
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
01630
01631
             continue;
01632
           /* Get indices... */
01633
           ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
01634
01635
```

```
iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01638
             /* Check indices... */
            if (ix < 0 || ix >= ctl->prof_nx ||
   iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01639
01640
01641
               continue;
01642
01643
             /\star Get total mass in grid cell... \star/
01644
            mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01645
01646
01647
          /* Extract profiles... */
          for (ix = 0; ix < ctl->prof_nx; ix++)
  for (iy = 0; iy < ctl->prof_ny; iy++)
01648
01649
01650
               if (obscount[ix][iy] > 0) {
01651
                 /* Write output... */
fprintf(out, "\n");
01652
01653
01654
                  /* Loop over altitudes... */
01656
                  for (iz = 0; iz < ctl->prof_nz; iz++) {
01657
01658
                    /* Set coordinates... */
                    z = ctl->prof_z0 + dz * (iz + 0.5);
lon = ctl->prof_lon0 + dlon * (ix + 0.5);
01659
01660
                   lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01661
01662
01663
                    /* Get meteorological data... */
01664
                    press = P(z);
                    intpol_met_time(met0, met1, t, press, lon, lat,
    NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
01665
01666
01667
                    /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
01668
01669
                   area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
 * cos(lat * M_PI / 180.);
01670
01671
                   mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
01672
01673
01674
                     /* Write output... */
                    fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g \n", tmean[ix][iy] / obscount[ix][iy],
01675
01676
                              z, lon, lat, press, temp, mmr, h2o, o3,
obsmean[ix][iy] / obscount[ix][iy]);
01677
01678
01679
01680
01681
01682
          /* Close file... */
          if (t == ctl->t_stop)
01683
01684
            fclose(out);
01685 }
```

Here is the call graph for this function:



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

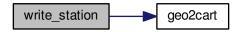
Write station data.

Definition at line 1689 of file libtrac.c.

```
{
01694
01695
        static FILE *out;
01696
01697
         static double rmax2, t0, t1, x0[3], x1[3];
01698
01699
        static int init, ip, iq;
01700
01701
         /* Init... */
01702
         if (!init) {
01703
           init = 1;
01704
01705
           /* Write info... */
01706
           printf("Write station data: %s\n", filename);
01707
           /* Create new file... */
01708
           if (!(out = fopen(filename, "w")))
01709
             ERRMSG("Cannot create file!");
01710
01711
01712
           /* Write header... */
01713
           fprintf(out,
                     "# $1 = time [s]\n"
"# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01714
01715
01716
           for (iq = 0; iq < ctl->nq; iq+)
fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
ctl->qnt_name[iq], ctl->qnt_unit[iq]);
fprintf(out, "\n");
01717
01718
01719
01720
01721
           /\star Set geolocation and search radius... \star/
01722
01723
           geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
01724
           rmax2 = gsl_pow_2(ctl->stat_r);
01725
01726
        /* Set time interval for output... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01727
01728
01729
01730
01731
         /* Loop over air parcels... */
01732
         for (ip = 0; ip < atm->np; ip++) {
01733
01734
           /* Check time... */
01735
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
01736
             continue;
01737
01738
           /\star Check station flag... \star/
01739
           if (ctl->qnt_stat >= 0)
01740
             if (atm->q[ctl->qnt_stat][ip])
01741
                continue:
01742
01743
           /* Get Cartesian coordinates... */
01744
           geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
01745
01746
           /\star Check horizontal distance... \star/
01747
           if (DIST2(x0, x1) > rmax2)
01748
             continue;
01749
01750
           /* Set station flag... */
01751
           if (ctl->qnt_stat >= 0)
01752
             atm->q[ctl->qnt_stat][ip] = 1;
01753
           /* Write data... */
fprintf(out, "%.2f %g %g %g",
01754
01755
01756
                    atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
           for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01757
01758
01759
01760
01761
           fprintf(out, "\n");
01762
01763
01764
         /* Close file... */
         if (t == ctl->t_stop)
01765
01766
           fclose(out);
01767 }
```

5.14 libtrac.h 113

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
00010
00011
        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
        GNU General Public License for more details.
00013
00014
        You should have received a copy of the GNU General Public License
00015
        along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>.
00016
00017
        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_statistics.h>
00040 #include <math.h>
00041 #include <netcdf.h>
00042 #include <omp.h>
00043 #include <stdio.h>
00044 #include <stdlib.h>
00045 #include <string.h>
00046 #include <time.h>
00047 #include <sys/time.h>
00048
00049 /* -----
00050 Macros...
00051
00052
00054 #define ALLOC(ptr, type, n)
00055 if((ptr=calloc((size_t)(n), sizeof(type)))==NULL)
          ERRMSG("Out of memory!");
00056
00057
00059 #define DIST(a, b) sgrt(DIST2(a, b))
00060
00062 #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]))
00063
00064
00066 #define DOTP(a, b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
00067
00069 #define ERRMSG(msg)
        printf("\nError (%s, %s, 1%d): %s\n\n",
00070
00071
                     FILE__,
                             __func__, __LINE__, msg);
00072
          exit(EXIT_FAILURE);
       }
00073
00074
00076 #define LIN(x0, y0, x1, y1, x)
00077 ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))
00078
00080 #define NC(cmd) {
        if((cmd)!=NC_NOERR)
00081
00082
             ERRMSG(nc_strerror(cmd));
00083
00084
```

```
00086 #define NORM(a) sqrt(DOTP(a, a))
00087
00089 #define PRINT(format, var)
00090 printf("Print (%s, %s, 1%d): %s= "format"\n",
00091
               __FILE__, __func__, __LINE__, #var, var);
00092
00094 #define P(z) (P0*exp(-(z)/H0))
00095
00097 #define TOK(line, tok, format, var) {
00098         if(((tok)=strtok((line), " \t"))) {
00099             if(sscanf(tok, format, &(var))!=1) continue;
00100         } else ERRMSG("Error while reading!");
00101
        }
00102
00104 #define Z(p) (H0*log(P0/(p)))
00105
00107 #define START_TIMER(id) timer(#id, id, 1)
00108
00110 #define STOP_TIMER(id) timer(#id, id, 2)
00113 #define PRINT_TIMER(id) timer(#id, id, 3)
00114
00115 /* -----
00116
        Constants...
00117
00118
00120 #define G0 9.80665
00121
00123 #define H0 7.0
00124
00126 #define P0 1013.25
00127
00129 #define RE 6367.421
00130
00131 /* -----
         Dimensions...
00132
00133
00134
00136 #define LEN 5000
00137
00139 #define NP 10000000
00140
00142 #define NO 5
00143
00145 #define EP 66
00146
00148 #define EX 361
00149
00151 #define EY 181
00152
00154 #define GX 720
00155
00157 #define GY 360
00158
00160 #define GZ 100
00161
00163 #define NTHREADS 128
00164
00166 #define NTIMER 20
00167
00168 /* -----
00169 Structs...
00170
00171
00173 typedef struct {
00174
00176
        int nq;
00177
00179
        char gnt name[NO][LEN];
00180
00182
        char qnt_unit[NQ][LEN];
00183
00185
        char qnt_format[NQ][LEN];
00186
00188
         int gnt m;
00189
00191
         int qnt_rho;
00192
00194
         int qnt_r;
00195
00197
         int qnt_ps;
00198
00200
         int qnt_t;
00201
00203
        int qnt_u;
00204
00206
        int qnt_v;
```

5.14 libtrac.h 115

```
00207
00209
        int qnt_w;
00210
00212
        int qnt_h2o;
00213
00215
        int qnt_o3;
00216
00218
        int qnt_theta;
00219
00221
        int qnt_stat;
00222
00224
        int direction:
00225
00227
        double t_start;
00228
00230
00231
        double t_stop;
00233
        double dt_mod;
00234
00236
        double dt_met;
00237
00240
        int isosurf;
00241
        char balloon[LEN];
00243
00244
00246
        double turb_dx_trop;
00247
00249
        double turb_dx_strat;
00250
00252
        double turb_dz_trop;
00253
        double turb_dz_strat;
00256
00258
        double turb_meso;
00259
        double tdec_trop;
00261
00262
        double tdec_strat;
00265
00267
        char atm_basename[LEN];
00268
00270
        char atm_gpfile[LEN];
00271
00273
        double atm_dt_out;
00274
00276
        char csi_basename[LEN];
00277
00279
        double csi_dt_out;
00280
00282
        char csi_obsfile[LEN];
00283
00285
        double csi_obsmin;
00286
00288
        double csi_modmin;
00289
00291
        int csi nz;
00292
00294
        double csi_z0;
00295
00297
        double csi_z1;
00298
00300
        int csi_nx;
00301
00303
        double csi_lon0;
00304
00306
        double csi_lon1;
00307
00309
        int csi_ny;
00310
        double csi_lat0;
00313
00315
        double csi_lat1;
00316
00318
        char grid_basename[LEN];
00319
        char grid_gpfile[LEN];
00322
00324
        double grid_dt_out;
00325
00327
        int grid_sparse;
00328
00330
        int grid_nz;
00331
00333
        double grid_z0;
00334
        double grid_z1;
00336
00337
```

```
00339
        int grid_nx;
00340
00342
        double grid_lon0;
00343
00345
        double grid_lon1;
00346
        int grid_ny;
00349
00351
        double grid_lat0;
00352
        double grid_lat1;
00354
00355
00357
        char prof_basename[LEN];
00358
00360
        char prof_obsfile[LEN];
00361
00363
        int prof_nz;
00364
00366
        double prof_z0;
00367
00369
        double prof_z1;
00370
00372
        int prof_nx;
00373
00375
        double prof_lon0;
00376
00378
        double prof_lon1;
00379
00381
        int prof_ny;
00382
00384
        double prof_lat0;
00385
00387
        double prof_lat1;
00388
00390
00391
        char stat_basename[LEN];
00393
        double stat_lon;
00394
00396
        double stat_lat;
00397
00399
        double stat_r;
00400
00401 } ctl_t;
00402
00404 typedef struct {
00405
00407
        int np;
00408
        double time[NP];
00410
00411
00413
        double p[NP];
00414
00416
        double lon[NP];
00417
00419
        double lat[NP];
00420
        double q[NQ][NP];
00423
00425
        double up[NP];
00426
00428
        double vp[NP];
00429
00431
        double wp[NP];
00432
00433 } atm_t;
00434
00436 typedef struct { 00437
00439
        double time;
00440
00442
        int nx;
00443
00445
        int ny;
00446
00448
        int np;
00449
00451
        double lon[EX];
00452
00454
        double lat[EY];
00455
00457
        double p[EP];
00458
00460
        double ps[EX][EY];
00461
00463
        float t[EX][EY][EP];
00464
00466
        float u[EX][EY][EP];
```

5.14 libtrac.h 117

```
00467
00469
        float v[EX][EY][EP];
00470
00472
       float w[EX][EY][EP];
00473
00475
       float h2o[EX][EY][EP];
00476
00478
        float o3[EX][EY][EP];
00479
00480 } met_t;
00481
00482 /* --
00483
         Functions...
00484
00485
00487 void cart2geo(
00488
       double *x,
00489
       double *z,
double *lon,
00490
00491
        double *lat);
00492
00494 double deg2dx(
       double dlon,
00495
00496
       double lat);
00497
00499 double deg2dy(
00500
       double dlat);
00501
00503 double dp2dz(
       double dp,
00504
00505
       double p);
00506
00508 double dx2deg(
00509
       double dx,
00510
       double lat);
00511
00513 double dy2deg(
      double dy);
00515
00517 double dz2dp(
00518
       double dz,
       double p);
00519
00520
00522 void geo2cart(
00523
       double z,
00524
        double lon,
00525
       double lat,
00526
       double *x);
00527
00529 void get_met(
       ctl_t * ctl,
00530
00531
        char *metbase,
00532
        double t,
       met_t * met0,
met_t * met1);
00533
00534
00535
00537 void get_met_help(
00538
       double t,
00539
        int direct,
00540
       char *metbase,
       double dt_met,
00541
00542
       char *filename);
00543
00545 void intpol_met_2d(
00546
       double array[EX][EY],
00547
       int ix,
00548
        int iy,
       double wx, double wy,
00549
00550
00551
        double *var);
00552
00554 void intpol_met_3d(
        float array[EX][EY][EP],
00555
00556
        int ip,
00557
        int ix,
00558
        int iy,
00559
        double wp,
00560
        double wx,
00561
        double wy,
00562
        double *var);
00563
00565 void intpol_met_space(
00566
       met_t * met,
00567
        double p,
00568
        double lon,
00569
        double lat,
00570
        double *ps,
```

```
00571
        double *t,
00572
        double *u,
00573
        double *v,
00574
        double *w,
00575
        double *h2o,
00576
        double *o3);
00577
00579 void intpol_met_time(
       met_t * met0,
met_t * met1,
00580
00581
        double ts,
00582
       double p, double lon,
00583
00584
00585
        double lat,
00586
        double *ps,
00587
        double *t,
00588
        double *u.
00589
        double *v,
00590
        double *w,
00591
        double *h2o,
00592
        double *o3);
00593
00595 void jsec2time(
00596
       double isec,
00597
        int *year,
00598
        int *mon,
00599
        int *day,
00600
       int *hour,
00601
        int *min,
00602
        int *sec,
00603
       double *remain);
00604
00606 int locate(
00607
       double *xx,
00608
       int n,
00609
        double x);
00610
00612 void read_atm(
00613
       const char *filename,
00614
        ctl_t * ctl,
00615
       atm_t * atm);
00616
00618 void read_ctl(
       const char *filename,
00619
00620
       int argc,
00621
        char *argv[],
00622
       ctl_t * ctl);
00623
00625 void read met (
00626 char *filename,
       met_t * met);
00627
00628
00630 void read_met_extrapolate(
00631
       met_t * met);
00632
00634 void read_met_help(
       int ncid,
00636
        char *varname,
00637
        char *varname2,
00638
        met_t * met,
00639
        int np,
       float dest[EX][EY][EP],
float scl);
00640
00641
00642
00644 void read_met_periodic(
00645
       met_t * met);
00646
00648 double scan_ctl(
00649
       const char *filename,
00650
        int argc,
00651
        char *argv[],
00652
        const char *varname,
       int arridx,
const char *defvalue,
char *value);
00653
00654
00655
00656
00658 void time2jsec(
00659
       int year,
00660
       int mon,
00661
        int day,
00662
        int hour,
00663
        int min,
00664
        int sec,
00665
        double remain,
00666
       double *jsec);
00667
00669 void timer(
```

```
const char *name,
00671
00672
         int mode);
00673
00674 /* Get tropopause pressure... */
00675 double tropopause(
00676 double t,
00677
         double lat);
00678
00680 void write_atm(
00681 const char *filename,

00682 ctl_t * ctl,

00683 atm_t * atm,
00684
00685
00687 void write_csi(
00688 const char *filename,
        ctl_t * ctl,
atm_t * atm,
00689
00690
00691
        double t);
00692
00694 void write_grid(
00695 const char *filename,
        ctl_t * ctl,
met_t * met0,
00696
00697
00698 met_t * met1,
00699 atm_t * atm,
00700 double t);
00701
00703 void write_prof(
00704 const char *filename,
        ctl_t * ctl,
met_t * met0,
00705
00706
        met_t * met1,
atm_t * atm,
double t);
00707
00708
00709
00710
00712 void write_station(
00713 const char *filename,
        ctl_t * ctl,
atm_t * atm,
00714
00715
00716 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

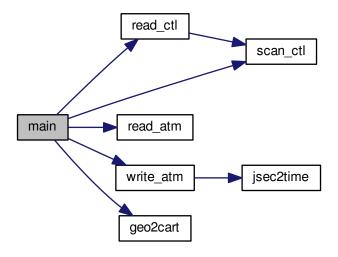
5.15.2.1 int main (int *argc*, char * *argv*[])

Definition at line 28 of file match.c.

```
00031
00032
        ctl_t ctl;
00033
00034
        atm t *atm1, *atm2, *atm3;
00035
        FILE *out;
00037
00038
        char filename[LEN];
00039
        double filter_dt, x1[3], x2[3], dh, dq[NQ], dv, 1h = 0, 1t = 0, 1v = 0;
00040
00041
00042
         int filter, ip1, ip2, iq, n;
00043
00044
         /* Allocate... */
        ALLOC(atm1, atm_t, 1);
ALLOC(atm2, atm_t, 1);
00045
00046
00047
         ALLOC(atm3, atm t, 1);
00048
00049
         /* Check arguments... */
00050
00051
           ERRMSG("Give parameters: <ctl> <atm_test> <atm_ref> <outfile>");
00052
00053
         /* Read control parameters... */
        read_ctl(argv[1], argc, argv, &ctl);
filter = (int) scan_ctl(argv[1], argc, argv, "FILTER", -1, "0", NULL);
filter_dt = scan_ctl(argv[1], argc, argv, "FILTER_DT", -1, "0", NULL);
00054
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: %s\n", argv[4]);
00064
         /* Create output file... */
00065
        if (!(out = fopen(argv[4], "w")))
00066
           ERRMSG("Cannot create file!");
00067
00068
00069
         /* Write header... */
        00070
00071
                  "# $2 = altitude [km] \n"
00072
         # $2 = drittedde [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>
00073
00074
00075
00076
                    ctl.qnt_unit[iq]);
        00077
00078
                  "# \$%d = vertical length of trajectory [km]\n"
00079
00080
                  "# $%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]);
00086
00087
         fprintf(out, "\n");
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++) {
00096
00097
             n = 0;
             atm2->p[ip1] = 0;
for (iq = 0; iq < ctl.nq; iq++)
00098
00099
              atm2->q[iq][ip1] = 0;
for (ip2 = 0; ip2 < atm2->np; ip2++)
00100
00101
               if (fabs(atm2->time[ip1] - atm2->time[ip2]) < filter_dt) {</pre>
00102
                 atm2->p[ip1] += atm3->p[ip2];
for (iq = 0; iq < ctl.nq; iq++)
00103
00104
00105
                   atm2->q[iq][ip1] += atm3->q[iq][ip2];
00106
00107
             atm2->p[ip1] /= n;
for (iq = 0; iq < ctl.nq; iq++)
00108
00109
               atm2->q[iq][ip1] /= n;
00110
00111
00112
00113
           /* Write filtered data... */
           sprintf(filename, "%s.filt", argv[3]);
00114
00115
           write_atm(filename, &ctl, atm2, 0);
00116
```

```
00118
         /* Loop over air parcels (reference data)... */
00119
         for (ip2 = 0; ip2 < atm2->np; ip2++) {
00120
00121
           /* Get trajectory length... */
           if (ip2 > 0) {
00122
00123
             geo2cart(0, atm2->lon[ip2 - 1], atm2->lat[ip2 - 1], x1);
00124
              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;
           dh = 0;
dv = 0;
00132
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
00140
00141
00142
00143
                / \star \ \texttt{Calculate deviations...} \ \star /
00144
                geo2cart(0, atm1->lon[ip1], atm1->lat[ip1], x1);
00145
                dh += DIST(x1, x2);
               dv += Z(atml->p[ip1]) - Z(atm2->p[ip2]);
for (iq = 0; iq < ctl.nq; iq++)</pre>
00146
00147
00148
                 dq[iq] += atm1->q[iq][ip1] - atm2->q[iq][ip2];
00149
00150
00151
           /* Write output... */
00152
00153
           if (n > 0)
             fprintf(out, "%.2f %.4f %.4f %.4f",
00154
00155
                      atm2->time[ip2], Z(atm2->p[ip2]),
00156
                      atm2->lon[ip2], atm2->lat[ip2]);
             for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl.qnt_format[iq], atm2->q[iq][ip2]);
00157
00158
00159
00160
00161
             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>
00162
00163
00164
00165
             fprintf(out, "\n");
00166
00167
          }
00168
00169
00170
        /* Close file... */
00171
        fclose(out);
00172
        /* 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 /*
00002
         This file is part of MPTRAC.
00003
00004
         MPTRAC is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by
00005
00006
         the Free Software Foundation, either version 3 of the License, or
00007
         (at your option) any later version.
80000
         MPTRAC is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
00009
00010
00011
         MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
         GNU General Public License for more details.
00013
         You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
00016
         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
         double filter_dt, x1[3], x2[3], dh, dq[NQ], dv, 1h = 0, 1t = 0, 1v = 0;
00040
00041
00042
         int filter, ip1, ip2, iq, n;
00043
00044
          /* Allocate... */
00045
00046
         ALLOC(atm1, atm_t, 1);
         ALLOC(atm2, atm_t, 1);
ALLOC(atm3, atm_t, 1);
00047
00048
         /* Check arguments... */
```

5.16 match.c 123

```
if (argc < 5)</pre>
00051
           ERRMSG("Give parameters: <ctl> <atm_test> <atm_ref> <outfile>");
00052
00053
        /\star Read control parameters... \star/
        read_ctl(argv[1], argc, argv, &ctl);
filter = (int) scan_ctl(argv[1], argc, argv, "FILTER", -1, "0", NULL);
00054
00055
        filter_dt = scan_ctl(argv[1], argc, argv, "FILTER_DT", -1, "0", NULL);
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: %s\n", argv[4]);
00064
        /* Create output file... */
if (!(out = fopen(argv[4], "w")))
00065
00066
          ERRMSG("Cannot create file!");
00067
00068
00069
        /* Write header... */
00070
        fprintf(out,
00071
                  "# $1 = time [s] \n"
                  "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00072
00073
        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,
00077
                  "# \$%d = trajectory time [s]\n"
00078
                  "# $%d = vertical length of trajectory [km]\n"
00079
                  "# $%d = horizontal length of trajectory [km]\n'
08000
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);
        00084
00085
00086
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... */
00096
           for (ip1 = 0; ip1 < atm2->np; ip1++) {
00097
            n = 0;
             atm2->p[ip1] = 0;
00098
             for (iq = 0; iq < ctl.nq; iq++)
atm2->q[iq][ip1] = 0;
00099
00100
00101
              for (ip2 = 0; ip2 < atm2->np; ip2++)
00102
               if (fabs(atm2->time[ip1] - atm2->time[ip2]) < filter_dt) {</pre>
                 atm2->p[ip1] += atm3->p[ip2];

for (iq = 0; iq < ctl.nq; iq++)

atm2->q[iq][ip1] += atm3->q[iq][ip2];
00103
00104
00105
00107
00108
              atm2->p[ip1] /= n;
             for (iq = 0; iq < ctl.nq; iq++)</pre>
00109
00110
               atm2->q[iq][ip1] /= n;
00111
00112
           /\star Write filtered data... \star/
00113
00114
           sprintf(filename, "%s.filt", argv[3]);
00115
          write_atm(filename, &ctl, atm2, 0);
00116
00117
00118
         /* Loop over air parcels (reference data)... */
         for (ip2 = 0; ip2 < atm2->np; ip2++) {
00119
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);
             lv += fabs(Z(atm2->p[ip2 - 1]) - Z(atm2->p[ip2]));
lt = fabs(atm2->time[ip2] - atm2->time[0]);
00126
00127
00128
00129
           /* Init... */
00130
00131
           n = 0;
00132
           dh = 0;
00133
           dv = 0;
           for (iq = 0; iq < ctl.nq; iq++)
  dq[iq] = 0;</pre>
00134
00135
00136
           geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
```

```
00138
           /* Find corresponding time step (test data)... */
           for (ipl = 0; ipl < atml->np; ipl++)
   if (fabs(atml->time[ip1] - atm2->time[ip2])
00139
00140
00141
                  < (filter ? filter_dt : 0.1)) {
00142
00143
               /* Calculate deviations... */
00144
                geo2cart(0, atm1->lon[ip1], atm1->lat[ip1], x1);
00145
                dh += DIST(x1, x2);
                dv += Z(atm1->p[ip1]) - Z(atm2->p[ip2]);
00146
               for (iq = 0; iq < ctl.nq; iq++)
    dq[iq] += atml->q[iq][ip1] - atm2->q[iq][ip2];
00147
00148
00149
               n++;
00150
00151
00152
           /* Write output... */
           if (n > 0) {
00153
             fprintf(out, "%.2f %.4f %.4f %.4f",
00154
                      atm2->time[ip2], Z(atm2->p[ip2]),
00155
                      atm2->lon[ip2], atm2->lat[ip2]);
00156
             for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");</pre>
00157
00158
               fprintf(out, ctl.qnt_format[iq], atm2->q[iq][ip2]);
00159
00160
00161
             fprintf(out, " %.2f %g %g %g %g", lt, lv, lh, dv / n, dh / n);
00162
             for (iq = 0; iq < ctl.nq; iq++) {
   fprintf(out, " ");</pre>
00163
00164
               fprintf(out, ctl.qnt_format[iq], dq[iq] / n);
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);
00177
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

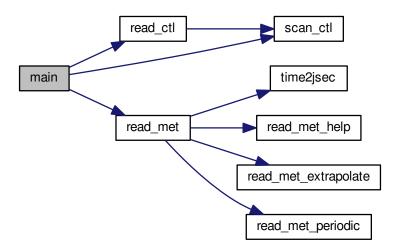
5.17.2.1 int main (int argc, char * argv[])

Definition at line 27 of file met_map.c.

```
00029
00030
00031
        ctl_t ctl;
00032
00033
        met t *met;
00034
        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
        if (argc < 4)
00047
           ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00048
00049
         /* Read control parameters... */
00050
        read_ctl(argv[1], argc, argv, &ctl);
        z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00051
00052
00053
         /* Loop over files... */
        for (i = 3; i < argc; i++) {
00054
00055
           /* Read meteorological data... */
00056
00057
           if (!(in = fopen(argv[i], "r")))
00058
             continue:
00059
           else
00060
             fclose(in);
00061
           read_met(argv[i], met);
00062
00063
           /\star Find nearest pressure level... \star/
           for (ip2 = 0; ip2 < met->np; ip2++) {
  dz = fabs(Z(met->p[ip2]) - z);
00064
00065
             if (dz < dzmin) {
00066
               dzmin = dz;
00067
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;
               tm[ix][iy] += met->t[ix][iy][ip];
um[ix][iy] += met->u[ix][iy][ip];
00076
00077
00078
               vm[ix][iy] += met->v[ix][iy][ip];
               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]);
if (!(out = fopen(argv[2], "w")))
00088
00089
           ERRMSG("Cannot create file!");
00090
00091
00092
         /* Write header... */
00093
         fprintf(out,
00094
                 "# $1 = time [s] \n"
                  "# $2 = altitude [km]\n"
00095
00096
                  "# $3 = longitude [deg]\n"
                  "# $4 = latitude [deg]\n"
00097
                  "# $5 = pressure [hPa]\n"
00098
00099
                  "# $6 = temperature [K] \n"
                  "# $7 = zonal wind [m/s] n"
00100
                  "# $8 = meridional wind [m/s]\n"
00101
                  "# $9 = vertical wind [hPa/s]\n"
00102
                  "# $10 = H20 volume mixing ratio [1]\n"
00103
                  "# $11 = 03 volume mixing ratio [1]\n
00104
00105
                  "# $12 = surface pressure [hPa]\n");
00106
00107
         /* Write data... */
        for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
  for (ix = 0; ix < met->nx; ix++)
00108
00109
00110
             00111
00112
                        timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
met->lon[ix] - 360.0, met->lat[iy], met->p[ip],
tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00113
00114
00115
```

```
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]);
00117
00118
               for (ix = 0; ix < met\rightarrownx; ix++)
00119
                 00120
00121
00122
00123
                                 met >folia; net >fat(iy), ne(ix)[iy], np[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
            /\star Close file... \star/
00131
           fclose(out);
00132
00133
            /* Free... */
00134
           free (met);
00135
00136
            return EXIT_SUCCESS;
00137 }
```

Here is the call graph for this function:



5.18 met map.c

```
00002
          This file is part of MPTRAC.
00003
         MPTRAC is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by
00004
00005
          the Free Software Foundation, either version 3 of the License, or
00006
00007
          (at your option) any later version.
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 */
00025 #include "libtrac.h"
```

5.18 met map.c 127

```
00026
00027 int main(
00028
        int argc,
00029
        char *argv[]) {
00030
00031
        ctl t ctl:
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
        if (argc < 4)
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... */
00054
        for (i = 3; i < argc; i++) {</pre>
00055
00056
           /* Read meteorological data... */
00057
           if (!(in = fopen(argv[i], "r")))
00058
             continue;
00059
           else
00060
            fclose(in);
          read_met(argv[i], met);
00061
00062
00063
           /* Find nearest pressure level... */
00064
           for (ip2 = 0; ip2 < met->np; ip2++) {
00065
             dz = fabs(Z(met->p[ip2]) - z);
00066
             if (dz < dzmin) {</pre>
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++) {
00073
00074
              timem[ix][iy] += met->time;
tm[ix][iy] += met->t[ix][iy][ip];
00075
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];
               h2om[ix][iy] += met->h2o[ix][iy][ip];
08000
              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]);
00089
        if (!(out = fopen(argv[2], "w")))
00090
          ERRMSG("Cannot create file!");
00091
00092
        /* Write header... */
00093
        fprintf(out,
                        = time [s]\n"
00094
                 "# $1
                        = altitude [km]\n"
00095
00096
                 "# $3 = longitude [deg] \n"
                 "# $4 = latitude [deg]\n"
00097
                 "# $5 = pressure [hPa]\n"
"# $6 = temperature [K]\n"
"# $7 = zonal wind [m/s]\n"
00098
00099
00100
                 "# $8 = meridional wind [m/s]\n"
00101
00102
                 "# $9 = vertical wind [hPa/s]\n"
                 00103
00104
                 "# $12 = surface pressure [hPa]\n");
00105
00106
00107
        /* Write data... */
        for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
00108
00109
           for (ix = 0; ix < met->nx; ix++)
00110
             00111
00112
```

```
timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
                timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
    met->lon[ix] - 360.0, met->lat[iy], met->p[ip],
    tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
    vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
    h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
    psm[ix][iy] / np[ix][iy]);

for (ix = 0; ix < met->nx; ix++)
00115
00116
00117
00118
00119
                    00120
00121
00122
00123
                                       met->lon[ix], met->lat[iy], met->p[ip],
                                      met=>lon(|x), met=>lat(|y), met=>p(|p),
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
              /* Close file... */
00130
00131
             fclose(out);
00132
00133
              /* Free... */
00134
             free(met);
00135
              return EXIT_SUCCESS;
00136
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[])

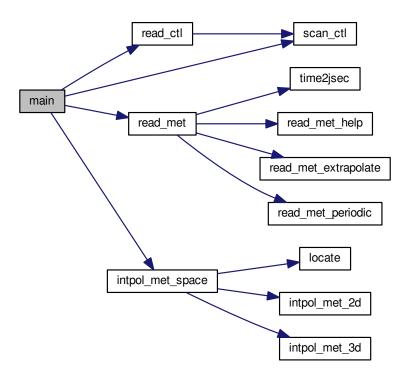
Definition at line 38 of file met_prof.c.

```
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], w, wm[NZ], h2o, h2om[NZ], o3, o3m[NZ];
00048
00049
00050
00051
00052
          static int i, iz, np[NZ];
00053
00054
00055
          /* Allocate... */
          ALLOC(met, met_t, 1);
00056
00057
          /* Check arguments... */
00058
          if (argc < 4)
```

```
ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00060
00061
            /* Read control parameters... */
           /* Read control parameters... */
read_ctl(argv[1], argc, argv, &ctl);
z0 = scan_ctl(argv[1], argc, argv, "Z0", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "Z1", -1, "60", NULL);
dz = scan_ctl(argv[1], argc, argv, "DZ", -1, "1", NULL);
lon0 = scan_ctl(argv[1], argc, argv, "LON0", -1, "0", NULL);
dlon = scan_ctl(argv[1], argc, argv, "LON1", -1, "0", NULL);
dlon = scan_ctl(argv[1], argc, argv, "DLON", -1, "1", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "LAT0", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "LAT1", -1, "0", NULL);
dlat = scan_ctl(argv[1], argc, argv, "DLAT", -1, "1", NULL);
00062
00063
00064
00066
00067
00068
00069
00070
00071
00072
00073
            /* Loop over input files... */
00074
            for (i = 3; i < argc; i++) {</pre>
00075
00076
               /* Read meteorological data... */
               if (!(in = fopen(argv[i], "r")))
00078
                 continue;
00079
00080
                 fclose(in);
               read_met(argv[i], met);
00081
00082
00083
               /* Average... */
               for (z = z0; z \le z1; z += dz) {
00085
                  iz = (int) ((z - z0) / dz);
00086
                  if (iz < 0 || iz > NZ)
00087
                     ERRMSG("Too many altitudes!");
                  for (lon = lon0; lon <= lon1; lon += dlon)
  for (lat = lat0; lat <= lat1; lat += dlat) {
    intpol_met_space(met, P(z), lon, lat, NULL,</pre>
00088
00089
00090
00091
                                                  &t, &u, &v, &w, &h2o, &o3);
00092
                        if (gsl_finite(t) && gsl_finite(u)
00093
                               && gsl\_finite(v) && gsl\_finite(w)) {
                           timem[iz] += met->time;
lonm[iz] += lon;
latm[iz] += lat;
00094
00095
00097
                           tm[iz] += t;
00098
                           um[iz] += u;
00099
                           vm[iz] += v;
                           wm[iz] += w;
00100
                           h2om[iz] += h2o;
00101
00102
                           o3m[iz] += o3;
                           np[iz]++;
00104
00105
                    }
          }
00106
00107
00108
            /* Normalize... */
00110
           for (z = z0; z <= z1; z += dz) {
00111
            iz = (int) ((z - z0) / dz);
             if (np[iz] > 0) {
  timem[iz] /= np[iz];
  lonm[iz] /= np[iz];
  latm[iz] /= np[iz];
00112
00113
00114
00116
                  tm[iz] /= np[iz];
                  um[iz] /= np[iz];
00117
                 vm[iz] /= np[iz];
wm[iz] /= np[iz];
00118
00119
                 h2om[iz] /= np[iz];
o3m[iz] /= np[iz];
00120
00121
00122
             } else {
00123
                 timem[iz] = GSL_NAN;
                 lonm[iz] = GSL_NAN;
latm[iz] = GSL_NAN;
00124
00125
                 tm[iz] = GSL_NAN;
um[iz] = GSL_NAN;
00126
00127
                  vm[iz] = GSL_NAN;
                  wm[iz] = GSL_NAN;
00129
00130
                  h2om[iz] = GSL_NAN;
                  o3m[iz] = GSL_NAN;
00131
00132
00133
00134
00135
            /* Create output file... */
           printf("Write meteorological data file: %s\n", argv[2]);
if (!(out = fopen(argv[2], "w")))
00136
00137
              ERRMSG("Cannot create file!");
00138
00139
00140
            /* Write header... */
00141
            fprintf(out,
00142
                        "# $1
                                  = time [s]\n"
                        "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n"
"# $4 = latitude [deg]\n"
00143
00144
00145
```

```
"# $5 = pressure [hPa]\n"
"# $6 = temperature [K]\n"
00147
                       "# $7 = zonal wind [m/s]\n"
00148
                       "# $8 = meridional wind [m/s]\n"
00149
                       "# $9 = vertical wind [hPa/s]\n"
00150
                       "# $10 = H2O volume mixing ratio [1]\n"
"# $11 = O3 volume mixing ratio [1]\n\n");
00151
00152
00153
          /* Write data... */
for (z = z0; z <= z1; z += dz) {
  iz = (int) ((z - z0) / dz);
  for int ((z - z0) / dz);
00154
00155
00156
             fprintf(out, "%.2f %g \n", timem[iz], z, lonm[iz], latm[iz], P(z), tm[iz], um[iz], vm[iz], wm[iz], h2om[iz], o3m[iz]);
00157
00158
00159
00160
00161
           /* Close file... */
00162
00163
           fclose(out);
00164
00165
           /* Free... */
00166
          free(met);
00167
00168
          return EXIT_SUCCESS;
00169 }
```

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
00006 the Free Software Foundation, either version 3 of the License, or
00007 (at your option) any later version.
```

5.20 met prof.c 131

```
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 /*
00028
           Dimensions...
00029
00030
00031 /* Maximum number of altitudes. */
00032 #define NZ 1000
00034 /*
00035
           Main...
00036
00037
00038 int main(
00039
          int argc,
          char *argv[]) {
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], w, wm[NZ], h2o, h2om[NZ], o3, o3m[NZ];
00048
00049
00050
00051
00052
          static int i, iz, np[NZ];
00053
           /* Allocate... */
00054
00055
          ALLOC(met, met_t, 1);
00056
00057
           /* Check arguments... */
00058
          if (argc < 4)</pre>
00059
             ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00060
          /* Read control parameters... */
read_ctl(argv[1], argc, argv, &ctl);
00061
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, "LON1", -1, "0", NULL);
dlon = scan_ctl(argv[1], argc, argv, "DLON", -1, "1", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "LAT0", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "LAT1", -1, "0", NULL);
00063
00064
00065
00066
00067
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
00076
              /* Read meteorological data...
00077
             if (!(in = fopen(argv[i], "r")))
00078
                continue;
00079
             else
08000
               fclose(in);
00081
             read_met(argv[i], met);
00082
00083
              /* Average... */
00084
              for (z = z0; z \le z1; z += dz) {
00085
                iz = (int) ((z - z0) / dz);
00086
                 if (iz < 0 || iz > NZ)
                   ERRMSG("Too many altitudes!");
00087
00088
                 for (lon = lon0; lon <= lon1; lon += dlon)</pre>
                   for (lat = lat0; lat <= lat1; lat += dlat)</pre>
00089
00090
                     intpol_met_space(met, P(z), lon, lat, NULL,
                      &t, &u, &v, &w, &h2o, &o3); if (gsl_finite(t) && gsl_finite(u)
00091
00092
                            && gsl_finite(v) && gsl_finite(w)) {
00093
                         timem[iz] += met->time;
00094
                        lonm[iz] += lon;
latm[iz] += lat;
00095
00096
00097
                         tm[iz] += t;
00098
                         um[iz] += u;
00099
                         vm[iz] += v;
00100
                         wm[iz] += w;
```

```
h2om[iz] += h2o;
00102
                     o3m[iz] += o3;
00103
                     np[iz]++;
00104
00105
00106
           }
00107
00108
         /* Normalize... */
for (z = z0; z <= z1; z += dz) {
  iz = (int) ((z - z0) / dz);
  if (np[iz] > 0) {
    timem[iz] /= np[iz];
00109
00110
00111
00112
00113
             lonm[iz] /= np[iz];
latm[iz] /= np[iz];
00114
00115
             tm[iz] /= np[iz];
um[iz] /= np[iz];
vm[iz] /= np[iz];
wm[iz] /= np[iz];
00116
00117
00118
00119
00120
              h2om[iz] /= np[iz];
00121
              o3m[iz] /= np[iz];
00122
              timem[iz] = GSL_NAN;
lonm[iz] = GSL_NAN;
latm[iz] = GSL_NAN;
00123
00124
00125
00126
              tm[iz] = GSL_NAN;
              um[iz] = GSL_NAN;
00127
              vm[iz] = GSL_NAN;
wm[iz] = GSL_NAN;
00128
00129
00130
              h2om[iz] = GSL_NAN;
              o3m[iz] = GSL_NAN;
00131
00132
            }
00133
00134
         /* Create output file... */
printf("Write meteorological data file: %s\n", argv[2]);
if (!(out = fopen(argv[2], "w")))
00135
00136
00137
            ERRMSG("Cannot create file!");
00139
00140
         /* Write header... */
         fprintf(out,
    "# $1 = time [s]\n"
    "# $2 = altitude [km]\n"
00141
00142
00143
                   "# $3 = longitude [deg]\n"
00144
                   "# $4 = latitude [deg]\n"
00146
                   "# $5 = pressure [hPa] \n"
00147
                   "# $6 = temperature [K] n"
                   "# $7 = zonal wind [m/s] \n"
00148
                   "# $8 = meridional wind [m/s]\n"
00149
                   "# $9 = vertical wind [hPa/s]\n"
00150
                   "# $10 = H20 volume mixing ratio [1]\n"
00151
00152
                   "# $11 = 03 volume mixing ratio [1]\n\n");
00153
        00154
00155
00156
00158
                     timem[iz], z, lonm[iz], latm[iz], P(z),
00159
                     tm[iz], um[iz], vm[iz], wm[iz], h2om[iz], o3m[iz]);
00160
00161
         /* Close file... */
00162
00163
         fclose(out);
00164
00165
         /* Free... */
00166
         free(met);
00167
00168
         return EXIT SUCCESS:
00169 }
```

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

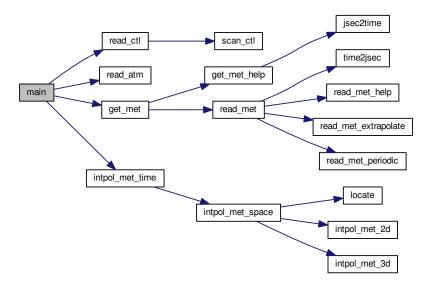
5.21.2.1 int main (int argc, char * argv[])

Definition at line 31 of file met_sample.c.

```
00033
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
00049
          ERRMSG("Give parameters: <ctl> <metbase> <atm_in> <sample.tab>");
00050
        /* Allocate... */
00051
00052
        ALLOC(atm, atm_t, 1);
       ALLOC(met0, met_t, 1);
00053
00054
        ALLOC(met1, met_t, 1);
00055
00056
       /* Read control parameters... */
00057
       read_ctl(argv[1], argc, argv, &ctl);
00058
00059
        /* Read atmospheric data... */
00060
        read_atm(argv[3], &ctl, atm);
00061
       /* Create output file... */
printf("Write meteorological data file: %s\n", argv[4]);
if (!(out = fopen(argv[4], "w")))
00062
00063
00064
          ERRMSG("Cannot create file!");
00065
00066
00067
        /* Write header... */
       fprintf(out,
00068
                "# $1 = time [s]\n"
"# $2 = altitude [km]\n"
00069
00070
00071
                "# $3 = longitude [deg] \n"
00072
                "# $4 = latitude [deg]\n"
00073
                "# $5 = pressure [hPa] \n"
                "# $6 = temperature [K]\n"
00074
                "# $7 = zonal wind [m/s]\n"
00075
00076
                "# $8 = meridional wind [m/s]\n"
                "# $9 = vertical wind [hPa/s]\n"
00077
                "# $10 = H20 volume mixing ratio [1]\n"
                "# $11 = 03 volume mixing ratio [1] \n\n");
00079
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
00087
          /* Interpolate meteorological data... */
          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
                  atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
00093
00094
                  atm->p[ip], t, u, v, w, h2o, o3);
00095
       }
00096
```

```
00097
        /* Close file... */
00098
        fclose(out);
00099
00100
        /* Free... */
00101
        free (atm);
00102
        free (met 0):
00103
        free (met1);
00104
00105
        return EXIT_SUCCESS;
00106 }
```

Here is the call graph for this function:



5.22 met_sample.c

```
00001 /*
          This file is part of MPTRAC.
00002
00003
          MPTRAC is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by
00004
00005
00006
          the Free Software Foundation, either version 3 of the License, or
00007
          (at your option) any later version.
80000
          MPTRAC is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00009
00010
00011
00012
          GNU General Public License for more details.
00013
00014
          You should have received a copy of the GNU General Public License
          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 /* --
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)
00049
         ERRMSG("Give parameters: <ctl> <metbase> <atm_in> <sample.tab>");
00050
00051
00052
        ALLOC(atm, atm_t, 1);
00053
        ALLOC(met0, met_t, 1);
00054
        ALLOC(met1, met_t, 1);
00055
00056
        /* Read control parameters... */
00057
       read_ctl(argv[1], argc, argv, &ctl);
00058
00059
       /* Read atmospheric data... */
00060
       read_atm(argv[3], &ctl, atm);
00061
00062
       /* Create output file... */
00063
       printf("Write meteorological data file: %s\n", argv[4]);
00064
       if (!(out = fopen(argv[4], "w")))
00065
         ERRMSG("Cannot create file!");
00066
00067
       /* Write header... */
00068
       fprintf(out,
00069
                "# $1 = time [s] \n"
                "# $2 = altitude [km]\n"
00070
                "# $3 = longitude [deg] \n"
00071
                "# $4 = latitude [deg]\n"
00072
00073
                "# $5 = pressure [hPa]\n"
00074
                "# $6 = temperature [K]\n"
00075
                "# $7 = zonal wind [m/s]\n"
00076
                "# $8 = meridional wind [m/s]\n"
                "# $9 = vertical wind [hPa/s]\n"
00077
                "# $10 = H2O volume mixing ratio [1]\n"
"# $11 = O3 volume mixing ratio [1]\n\n");
00078
00079
08000
        /* 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
00087
          /* Interpolate meteorological data... */
          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
                 atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
00094
                 atm->p[ip], t, u, v, w, h2o, o3);
00095
00096
        /* Close file... */
00097
00098
       fclose(out);
00099
00100
       /* Free... */
00101
       free (atm);
00102
        free (met0);
00103
        free (met1):
00104
00105
        return EXIT_SUCCESS;
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

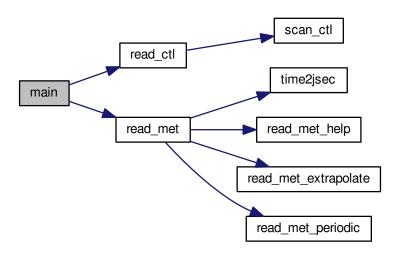
5.23.2.1 int main (int argc, char * argv[])

Definition at line 27 of file met zm.c.

```
00029
00031
         ctl_t ctl;
00032
00033
         met_t *met;
00034
00035
         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, iv, np[EP][EY];
00043
00044
         /* Allocate... */
00045
         ALLOC(met, met_t, 1);
00046
00047
         /* Check arguments... */
00048
         if (argc < 4)
           ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00049
00050
00051
         /\star Read control parameters... \star/
00052
         read_ctl(argv[1], argc, argv, &ctl);
00053
00054
         /* Loop over files... */
00055
         for (i = 3; i < argc; i++) {
00056
00057
            /* Read meteorological data... */
00058
           if (!(in = fopen(argv[i], "r")))
00059
             continue;
00060
           else
00061
             fclose(in);
00062
           read_met(argv[i], met);
00063
00064
           /* Average data... */
           for (ix = 0; ix < met->nx; ix++)
  for (iy = 0; iy < met->ny; iy++)
    for (ip = 0; ip < met->np; ip++) {
00065
00066
00067
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
                  \label{limits} $$ 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];
00075
                  h2om[ip][iy] += met->h2o[ix][iy][ip];
00076
                  o3m[ip][iy] += met->o3[ix][iy][ip];
                  psm[ip][iy] += met->ps[ix][iy];
00077
00078
                  tm2[ip][iy] += gsl_pow_2(met->t[ix][iy][ip]);
                  um2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip]);
00079
                  vm2[ip][iy] += gsl_pow_2(met->v[ix][iy][ip]);
vhm2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip])
00080
00081
00082
                    + gsl_pow_2 (met->v[ix][iy][ip]);
00083
                  wm2[ip][iy] += gsl_pow_2(met->w[ix][iy][ip]);
                  00084
00085
00086
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")))
```

```
00094
          ERRMSG("Cannot create file!");
00095
00096
        /* Write header... */
00097
       fprintf(out,
                "# $1
00098
                      = time [s]\n"
               "# $2 = altitude [km]\n"
"# $3 = latitude [deg]\n"
00099
00101
               "# $4 = temperature mean [K]\n"
00102
               "# $5 = temperature standard deviation [K]\n"
                "# $6 = zonal wind mean [m/s]\n"
00103
               "# $7 = zonal wind standard deviation [m/s]\n"
00104
               "# $8 = meridional wind mean [m/s]\n"
00105
                      = meridional wind standard deviation [m/s]\n"
00106
               "# $9
00107
               "# $10 = horizontal wind mean [m/s]\n"
00108
               "# $11 = horizontal wind standard deviation [m/s]\n"
               "# $12 = vertical wind mean [hPa/s]\n"
"# $13 = vertical wind standard deviation [hPa/s]\n"
00109
00110
               "# $14 = H20 vmr mean [1]\n"
"# $15 = H20 vmr standard deviation [1]\n"
00111
00112
               "# $16 = 03 vmr mean [1]\n"
00113
00114
               "# $17 = 03 vmr standard deviation [1]\n"
                "# $18 = surface pressure mean [hPa] \n"
00115
               "# $19 = surface pressure standard deviation [hPa] \n");
00116
00117
00118
       /* Write data... */
       for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
00119
00120
         00121
00122
00123
00124
00125
00126
                        gsl_pow_2(tm[ip][iy] / np[ip][iy])),
00127
                   um[ip][iy] / np[ip][iy],
00128
                   00129
00130
00131
00132
00133
00134
00135
00136
00137
00138
00139
                        gsl_pow_2(wm[ip][iy] / np[ip][iy])),
                   00140
00141
00142
00143
00144
00145
00146
                   sqrt(psm2[ip][iy] / np[ip][iy] -
00147
                        gsl_pow_2(psm[ip][iy] / np[ip][iy])));
00148
00149
        /* Close file... */
00151
00152
       fclose(out);
00153
00154
       /* Free... */
00155
       free (met);
00156
       return EXIT_SUCCESS;
00158 }
```

Here is the call graph for this function:



5.24 met zm.c

```
00001 /*
00002
                       This file is part of MPTRAC.
00003
                      MPTRAC is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by
00004
00005
                       the Free Software Foundation, either version 3 of the License, or
00006
00007
                       (at your option) any later version.
80000
00009
                       \ensuremath{\mathsf{MPTRAC}} is distributed in the hope that it will be useful,
                       but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
00012
                       GNU General Public License for more details.
00013
00014
                       You should have received a copy of the GNU General Public License
00015
                       along with MPTRAC. If not, see <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 int main(
00028
                      int argc,
                      char *argv[]) {
00029
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], vm[EP][EY], vm[EP][EY], wm[EP][EY], h2om[EP][EY], o3m[EP][EY], psm2[EP][EY], tm2[EP][EY], um2[EP][EY], vm2[EP][EY], vm2[EP][EY], vm2[EP][EY], vm2[EP][EY], o3m2[EP][EY], o3m2[EP][EY], vm2[EP][EY], vm2[EY], vm2[EY][EY], vm2[EY][EY], vm2[EY], vm2[EY][EY], vm2[EY][EY]
00037
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> <map.tab> <met0> [ <met1> ... ]");
00050
```

5.24 met zm.c 139

```
/* Read control parameters... */
00052
         read_ctl(argv[1], argc, argv, &ctl);
00053
00054
         /* Loop over files... ∗/
00055
         for (i = 3; i < argc; i++) {
00056
            /* Read meteorological data... */
00058
            if (!(in = fopen(argv[i], "r")))
00059
              continue;
00060
           else
00061
             fclose(in);
00062
           read_met(argv[i], met);
00063
00064
            /* Average data... */
00065
            for (ix = 0; ix < met->nx; ix++)
              for (iy = 0; iy < met->ny; iy++)
  for (ip = 0; ip < met->np; ip++) {
    timem[ip][iy] += met->time;
00066
00067
00068
                   tm[ip][iy] += met->t[ix][iy][ip];
00070
                   um[ip][iy] += met->u[ix][iy][ip];
00071
                   vm[ip][iy] += met->v[ix][iy][ip];
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];
00075
                   h2om[ip][iy] += met->h2o[ix][iy][ip];
                   o3m[ip][iy] += met->o3[ix][iy][ip];
00076
00077
                   psm[ip][iy] += met->ps[ix][iy];
00078
                   tm2[ip][iy] += gsl_pow_2(met->t[ix][iy][ip]);
00079
                   um2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip]);
                  vm2[ip][iy] += gsl_pow_2(met->v[ix][iy][ip]);
vhm2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip])
+ gsl_pow_2(met->v[ix][iy][ip]);
00080
00081
00082
00083
                   wm2[ip][iy] += gsl_pow_2(met->w[ix][iy][ip]);
00084
                   \label{eq:h2om2[ip][iy] += gsl_pow_2(met->h2o[ix][iy][ip]);} h2om2[ip][iy] += gsl_pow_2(met->h2o[ix][iy][ip]);
                   o3m2[ip][iy] += gsl_pow_2(met->o3[ix][iy][ip]);
psm2[ip][iy] += gsl_pow_2(met->ps[ix][iy]);
00085
00086
00087
                   np[ip][iy]++;
00089
00090
00091
         /* Create output file... */
         printf("Write meteorological data file: %s\n", argv[2]);
00092
         if (!(out = fopen(argv[2], "w")))
00093
           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
00101
                          = temperature mean [K]\n"
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'
00105
                   "# $9 = meridional wind standard deviation [m/s]\n"
00106
                   "# $10 = horizontal wind mean [m/s]\n"
00108
                   "# $11 = horizontal wind standard deviation [m/s]\n"
00109
                   "# $12 = vertical wind mean [hPa/s]\n"
                   "# $13 = vertical wind standard deviation [hPa/s]\n"
00110
                   "# $14 = H20 \text{ vmr mean } [1] \n"
00111
                   "# $15 = H2O vmr standard deviation [1]\n"
00112
00113
                   "# $16 = 03 \text{ vmr mean } [1] \n"
                   "# $17 = 03 vmr standard deviation [1]\n"
00114
00115
                   "# $18 = surface pressure mean [hPa] \n"
00116
                   "# $19 = surface pressure standard deviation [hPa] \n");
00117
00118
         /* Write data... */
        /* Write data... */
for (iy = 0; iy < met->ny; iy++) {
   fprintf(out, "\n");
   for (ip = 0; ip < met->np; ip++)
     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g"
00119
00120
00121
00122
                       (out, "8.21 *g *g *g *g *g *g *g *g *g *g
" *g *g *g *g *g *g *g,",
timem[ip][iy] / np[ip][iy], Z(met->p[ip]), met->lat[iy],
tm[ip][iy] / np[ip][iy],
sqrt(tm2[ip][iy] / np[ip][iy] -
00123
00124
00125
00126
00127
                             gsl_pow_2(tm[ip][iy] / np[ip][iy])),
00128
                        um[ip][iy] / np[ip][iy],
                       00129
00130
00131
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] -
00140
00141
                      gsl_pow_2(h2om[ip][iy] / np[ip][iy])),
o3m[ip][iy] / np[ip][iy],
sqrt(o3m2[ip][iy] / np[ip][iy] -
00142
00143
00144
00145
                             gsl_pow_2(o3m[ip][iy] / np[ip][iy])),
                      00146
00147
00148
00149
00150
00151
        /* Close file... */
00152
        fclose(out);
00153
        /* Free... */
00154
00155
        free (met);
00156
00157
         return EXIT_SUCCESS;
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[])

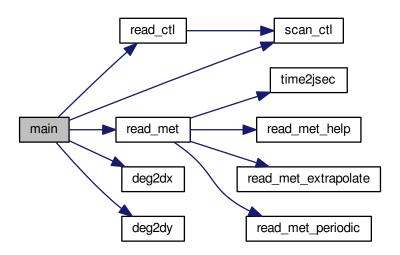
Definition at line 8 of file smago.c.

```
00010
00011
00012
         ctl_t ctl;
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
```

```
/* Read meteorological data... */
00034
        read_met(argv[3], met);
00035
        /\star Find nearest pressure level... \star/
00036
        for (ip2 = 0; ip2 < met->np; ip2++) {
00037
          dz = fabs(Z(met->p[ip2]) - z);
00038
          if (dz < dzmin) {
00040
            dzmin = dz;
             ip = ip2;
00041
00042
        }
00043
00044
00045
        /* Write info... */
00046
        printf("Analyze %g hPa...\n", met->p[ip]);
00047
00048
        /* Calculate horizontal diffusion coefficients... */
        for (ix = 1; ix < met->nx - 1; ix++)
  for (iy = 1; iy < met->ny - 1; iy++) {
    t = 0.5 * ((met->u[ix + 1][iy][ip] - met->u[ix - 1][iy][ip])
00049
00050
00051
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])
00055
00056
                        / (1000. * deg2dy(met->lat[iy + 1] - met->lat[iy
00058
                        + (met->v[ix + 1][iy][ip] - met->v[ix - 1][iy][ip])
00059
                        / (1000. *
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
00063
            if (fabs(met->lat[iy]) > 80)
00064
              ls2 *= (90. - fabs(met->lat[iy])) / 10.;
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
08000
00081
        /* Write data... */
        for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
00082
00083
00084
           for (ix = 0; ix < met->nx; ix++)
            00085
00086
00087
                       met->u[ix][iy][ip], met->v[ix][iy][ip], k[ix][iy]);
00089
          for (ix = 0; ix < met->nx; ix++)
00090
            if (met->lon[ix] <= 180 && met->lon[ix] > 0)
              00091
00092
00093
                       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);
00102
        return EXIT_SUCCESS;
00103 }
```

Here is the call graph for this function:



5.26 smago.c

```
00001
00006 #include "libtrac.h"
00007
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... */
         if (argc < 4)
00026
           ERRMSG("Give parameters: <ctl> <map.tab> <met>");
00027
00028
         /* Read control parameters... */
read_ctl(argv[1], argc, argv, &ctl);
z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00029
00030
00031
00032
00033
          /\star Read meteorological data... \star/
00034
         read_met(argv[3], met);
00035
00036
          /* Find nearest pressure level... */
         /* Find hearest pressure level... */
for (ip2 = 0; ip2 < met->np; ip2++) {
    dz = fabs(Z(met->p[ip2]) - z);
    if (dz < dzmin) {
        dzmin = dz;
    }
}</pre>
00037
00038
00039
00040
00041
              ip = ip2;
00042
            }
00043
00044
00045
         /\star Write info... \star/
00046
         printf("Analyze %g hPa...\n", met->p[ip]);
00047
00048
         /* Calculate horizontal diffusion coefficients... */
         for (ix = 1; ix < met->nx - 1; ix++)
```

```
for (iy = 1; iy < met->ny - 1; iy++) {
00051
           t = 0.5 * ((met > u[ix + 1][iy][ip] - met > u[ix - 1][iy][ip])
00052
                       / (1000. *
00053
                          deg2dx(met->lon[ix + 1] - met->lon[ix - 1], met->
     lat[iy]))
00054
                       - (met->v[ix][iy + 1][ip] - met->v[ix][iy - 1][ip])
            / (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])
00055
00056
00057
                       / (1000. * deg2dy(met->lat[iy + 1] - met->lat[iy
                                                                          - 1]))
00058
                       + (met->v[ix + 1][iy][ip] - met->v[ix - 1][iy][ip])
00059
                       / (1000. *
                          deg2dx(met->lon[ix + 1] - met->lon[ix - 1],
00060
                                 met->lat[iy])));
00061
00062
            ls2 = gsl_pow_2(c * 500. * deg2dy(met->lat[iy + 1] - met->lat[iy - 1]));
00063
            if (fabs(met->lat[iy]) > 80)
00064
              1s2 \star = (90. - fabs(met->lat[iy])) / 10.;
            k[ix][iy] = 1s2 * sqrt(2.0 * (gsl_pow_2(t) + gsl_pow_2(s)));
00065
00066
00067
00068
       /* Create output file... */
00069
       printf("Write data file: %s\n", argv[2]);
00070
        if (!(out = fopen(argv[2], "w")))
        ERRMSG("Cannot create file!");
00071
00072
00073
        /* Write header... */
00074
       fprintf(out,
00075
                "# $1 = longitude [deg] \n"
                "# $2 = latitude [deg] \n"
00076
00077
                "# $3 = zonal wind [m/s] \n"
                "# $4 = meridional wind [m/s]\n"
00078
00079
                "# $5 = horizontal diffusivity [m^2/s]\n");
08000
00081
       /* Write data... */
        for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
  for (ix = 0; ix < met->nx; ix++)
00082
00083
00084
           00085
00086
00087
88000
                      met->u[ix][iy][ip], met->v[ix][iy][ip], k[ix][iy]);
00089
          for (ix = 0; ix < met->nx; ix++)
          00090
00091
00092
00093
                      met->u[ix][iy][ip], met->v[ix][iy][ip], k[ix][iy]);
00094
00095
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

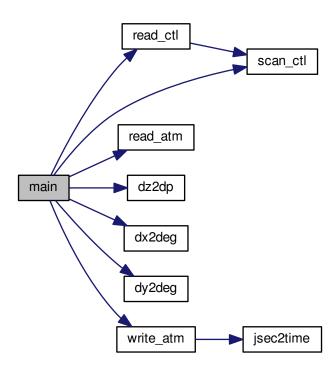
5.27.2.1 int main (int argc, char * argv[])

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_Z0", -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);
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... */
          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
          /* Get total and maximum mass... */
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
00107
            /\star Set horizontal position... \star/
           if (lon1 > lon0 && lat1 > lat0) {
00108
             atm2->lon(atm2->np) = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
atm2->lat(atm2->np) = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
00109
00110
00111
00112
            atm2 \rightarrow lon[atm2 \rightarrow np] = atm \rightarrow lon[ip]
00113
                + gsl_ran_gaussian_ziggurat(rng, dx2deg(dx, atm->lat[ip]) / 2.3548);
00114
             atm2->lat[atm2->np] = atm->lat[ip]
                + gsl_ran_gaussian_ziggurat(rng, dy2deg(dx) / 2.3548);
00115
00116
00117
00118
            /* Copy quantities... */
00119
           for (iq = 0; iq < ctl.nq; iq++)
            atm2->q[iq][atm2->np] = atm->q[iq][ip];
00120
00121
           /* Adjust mass... */
if (ctl.qnt_m >= 0)
00122
00123
00124
             atm2->q[ct1.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

```
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);
dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);
00051
00052
00053
00054
00055
00056
          dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);
00057
          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... */
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
           /\star Set time... \star/
00093
00094
           if (t1 > t0)
             atm2->time[atm2->np] = t0 + (t1 - t0) * gsl_rng_uniform_pos(rng);
00095
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
00107
           /* Set horizontal position... */
if (lon1 > lon0 && lat1 > lat0) {
00108
            atm2->lon[atm2->np] = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
atm2->lat[atm2->np] = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
00109
00110
00111
00112
             atm2 \rightarrow lon[atm2 \rightarrow np] = atm \rightarrow 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... */
00119
           for (iq = 0; iq < ctl.nq; iq++)</pre>
00120
             atm2->q[iq][atm2->np] = atm->q[iq][ip];
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
00138
        return EXIT_SUCCESS;
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.

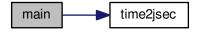
5.29.2 Function Documentation

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
00004
        MPTRAC is free software: you can redistribute it and/or modify
00005
         it under the terms of the GNU General Public License as published by
00006
        the Free Software Foundation, either version 3 of the License, or
00007
         (at your option) any later version.
80000
00009
        MPTRAC is distributed in the hope that it will be useful,
        but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
00012
        GNU General Public License for more details.
00013
00014
        You should have received a copy of the GNU General Public License
00015
        along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/>.
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.

5.31.2 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
00402
         /\star Set inital and final time... \star/
00403
         if (ctl->direction == 1)
00404
           if (ctl->t_start < -1e99)
           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);
00405
00406
00407
         } else if (ctl->direction == -1) {
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 }
```

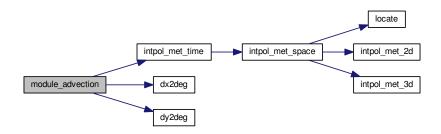
5.31.2.2 void module_advection (met_t * met0, met_t * met1, atm_t * atm, int ip, double dt)

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, \\ & vv[0], \, & vv[1], \, & vv[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
           atm->p[ip] += dt * v[2];
00450
00451 }
```

Here is the call graph for this function:



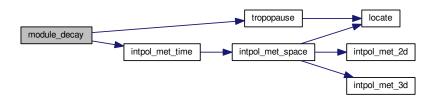
5.31.2.3 void module_decay (ctl_t * ctl, met_t * met0, met_t * met1, atm_t * atm, int ip, double dt)

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.

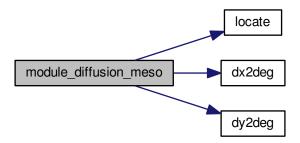
```
00505
00506
         double r, rs, u[16], v[16], w[16], usig, vsig, wsig;
00507
00508
         int ix, iv, iz;
00509
00510
          /* Calculate mesoscale velocity fluctuations... */
00511
          if (ctl->turb_meso > 0) {
00512
00513
            /* Get indices... */
00514
           ix = locate(met0->lon, 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
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];
00529
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
00541
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];
u[9] = met1->u[ix + 1][iy][iz];
00552
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[13] = 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) [iy + 1] [iz];
w(12) = met1->w(ix) [iy] [iz + 1];
00572
00573
00574
00575
            w[13] = met1->w[ix + 1][iy][iz + 1];
            w[14] = met1->w[ix][iy + 1][iz + 1];
w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00576
00577
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/
00589
            atm->up[ip]
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,
00594
                                                                             ctl->turb_meso * vsig);
00595
            atm->wp[ip] =
00596
              r * atm->wp[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
                                                                            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 }
```

Here is the call graph for this function:



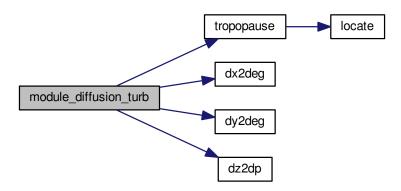
5.31.2.5 void module diffusion turb (ctl t*ctl, atm t*atm, int ip, double dt, gsl rng *rng)

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
00625
        else if (atm->p[ip] < p1)
          w = 0;
00626
00627
        else
          w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00628
00629
        /* Set diffusivitiy... */
        dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
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)

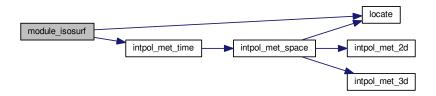
Force air parcels to stay on isosurface.

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
00672
         /* 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],
```

```
00692
                                   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, LEN, 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
              <u>if</u> (n < 1)
00724
                ERRMSG("Could not read any data!");
00725
00726
              /* Close file... */
00727
              fclose(in);
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 {
             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:



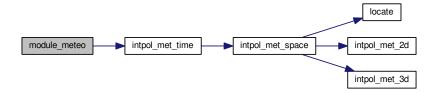
```
5.31.2.7 void module_meteo ( ctl t * ctl, met t * met0, met t * met1, atm t * atm, int ip )
```

Interpolate meteorological data for air parcel positions.

Definition at line 768 of file trac.c.

```
00773
                 {
00774
00775
        /* Interpolate surface pressure... */
00776
        if (ctl->qnt_ps >= 0)
00777
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00778
                           atm->lat[ip], &atm->q[ctl->qnt_ps][ip], NULL,
NULL, NULL, NULL, NULL);
00779
00780
00781
        /* Interpolate temperature... */
00782
        if (ctl->qnt_t >= 0)
00783
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00784
                           atm->lat[ip], NULL, &atm->q[ctl->qnt_t][ip],
NULL, NULL, NULL, NULL, NULL);
00785
00787
        /* Interpolate zonal wind... */
00788
        if (ctl->qnt_u >= 0)
00789
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00790
                           atm->lat[ip], NULL, NULL, &atm->q[ctl->qnt_u][ip],
00791
                           NULL, NULL, NULL, NULL);
00792
00793
        /* Interpolate meridional wind... */
00794
        if (ctl->qnt_v >= 0)
00795
         intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00796
                           atm->lat[ip], NULL, NULL, NULL,
00797
                           &atm->q[ctl->qnt_v][ip], NULL, NULL, NULL);
00798
00799
        /* Interpolate vertical velocity... */
00800
        if (ctl->qnt_w >= 0)
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00801
      lon[ip],
00802
                           atm->lat[ip], NULL, NULL, NULL, NULL,
00803
                           &atm->q[ctl->qnt_w][ip], NULL, NULL);
00804
00805
        /* Interpolate water vapor vmr... */
        if (ctl->qnt_h2o >= 0)
00806
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00807
      lon[ip],
80800
                           atm->lat[ip], NULL, NULL, NULL, NULL, NULL,
00809
                           &atm->q[ctl->qnt_h2o][ip], NULL);
00810
00811
        /* Interpolate ozone... */
00812
        if (ctl->qnt_o3 >= 0)
00813
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00814
                           atm->lat[ip], NULL, NULL, NULL, NULL, NULL, NULL,
00815
                           &atm->q[ctl->qnt_o3][ip]);
00816
00817
        /* Calculate potential temperature... */
00818
        if (ctl->qnt_theta >= 0) {
00819
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
```

Here is the call graph for this function:



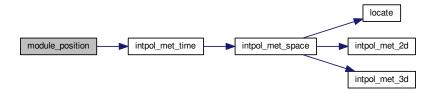
```
5.31.2.8 void module_position ( met_t * met0, met_t * met1, atm_t * atm, int ip )
```

Check position of air parcels.

Definition at line 828 of file trac.c.

```
00832
                        {
00833
00834
           double ps;
00835
00836
           /* Calculate modulo... */
00837
           atm \rightarrow lon[ip] = fmod(atm \rightarrow lon[ip], 360);
00838
           atm \rightarrow lat[ip] = fmod(atm \rightarrow lat[ip], 360);
00839
           /* Check latitude... */
while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
   if (atm->lat[ip] > 90) {
     atm->lat[ip] = 180 - atm->lat[ip];
00840
00841
00842
00843
00844
                 atm->lon[ip] += 180;
00845
              if (atm->lat[ip] < -90) {
  atm->lat[ip] = -180 - atm->lat[ip];
  atm->lon[ip] += 180;
00846
00847
00848
00849
              }
00850
00851
           /* Check longitude... */
while (atm->lon[ip] < -180)
  atm->lon[ip] += 360;
while (atm->lon[ip] >= 180)
  atm->lon[ip] -= 360;
00852
00853
00854
00855
00856
00857
00858
           /* Get surface pressure... */
           intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
    atm->lon[ip], atm->lat[ip], &ps, NULL,
00859
00860
                                    NULL, NULL, NULL, NULL, NULL);
00861
00862
00863
            /* Check pressure... */
           if (atm->p[ip] > ps)
  atm->p[ip] = ps;
else if (atm->p[ip] < met0->p[met0->np - 1])
00864
00865
00866
              atm->p[ip] = met0->p[met0->np - 1];
00867
00868 }
```

Here is the call graph for this function:



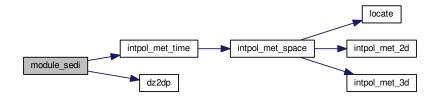
5.31.2.9 void module_sedi ($ctl_t * ctl$, $met_t * met0$, $met_t * met1$, $atm_t * atm$, int ip, double dt)

Calculate sedimentation of air parcels.

Definition at line 872 of file trac.c.

```
00878
00879
00880
         /* Coefficients for Cunningham slip-flow correction (Kasten, 1968): */
00881
        const double A = 1.249, B = 0.42, C = 0.87;
00882
        /* Specific gas constant for dry air [J/(kg\ K)]: */const double R = 287.058;
00883
00884
00885
00886
         /\star Average mass of an air molecule [kg/molec]: \star/
00887
        const double m = 4.8096e-26;
00888
00889
        double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
00890
00891
        /* Check if parameters are available... */
00892
        if (ctl->qnt_r < 0 || ctl->qnt_rho < 0)</pre>
00893
00894
00895
         /* Convert units... */
00896
        p = 100 * atm->p[ip];
00897
        r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
        rho_p = atm->q[ctl->qnt_rho][ip];
00898
00899
00900
         /* Get temperature... */
00901
        intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00902
                          atm->lat[ip], NULL, &T, NULL, NULL, NULL, NULL, NULL);
00903
00904
         /* Density of dry air... */
00905
        rho = p / (R * T);
00906
00907
        /* Dynamic viscosity of air... */
eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
00908
00909
        /* Thermal velocity of an air molecule... */ v = sqrt(8 * GSL_CONST_MKSA_BOLTZMANN * T / (M_PI * m));
00910
00911
00912
00913
         /\star Mean free path of an air molecule... \star/
00914
        lambda = 2 * eta / (rho * v);
00915
00916
         /* Knudsen number for air... */
00917
        K = lambda / r_p;
00918
00919
         /\star Cunningham slip-flow correction... \star/
00920
        G = 1 + K * (A + B * exp(-C / K));
00921
00922
         /* Sedimentation (fall) velocity... */
00923
         v_p =
00924
           2. * gsl_pow_2(r_p) * (rho_p -
00925
                                    rho) * GSL_CONST_MKSA_GRAV_ACCEL / (9. * eta) * G;
00926
00927
        /* Calculate pressure change... */
00928
        atm > p[ip] += dz2dp(v_p * dt / 1000., atm > p[ip]);
00929 }
```

Here is the call graph for this function:



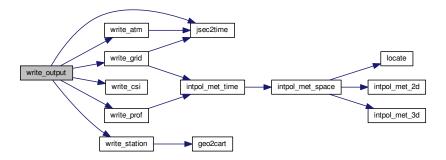
5.31.2.10 void write_output (const char * dirname, ctl_t * ctl, met_t * met0, met_t * met1, atm_t * atm, double t)

Write simulation output.

Definition at line 933 of file trac.c.

```
00939
                        {
00940
00941
          char filename[LEN]:
00942
00943
          double r;
00944
00945
          int year, mon, day, hour, min, sec;
00946
00947
          /* Get time... */
00948
          jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
00949
          /* Write atmospheric data... */
if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
    sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.tab",
00950
00951
00952
00953
                        dirname, ctl->atm_basename, year, mon, day, hour, min);
00954
             write_atm(filename, ctl, atm, t);
00955
00956
00957
           /* Write gridded data...
          if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
    sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d_tab",
00958
00959
             dirname, ctl-ygrid_basename, year, mon, day, hour, min); write_grid(filename, ctl, met0, met1, atm, t);
00960
00961
00962
00963
00964
          /* Write CSI data... */
          if (ctl->csi_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
00965
00966
            write_csi(filename, ctl, atm, t);
00967
00968
00969
          /* Write profile data... */
if (ctl->prof_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
00970
00971
00972
00973
             write_prof(filename, ctl, met0, met1, atm, t);
00974
00975
00976
          /* Write station data...
          /^ wille station data... ^/
if (ctl->stat_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
00977
00978
00979
             write_station(filename, ctl, atm, t);
00980
00981 }
```

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.

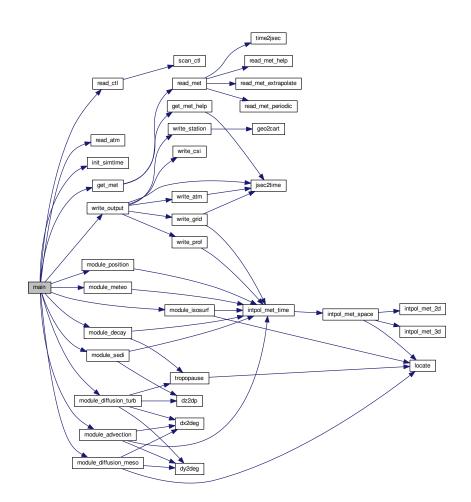
```
00162
00163
00164
        ctl_t ctl;
00165
00166
        atm_t *atm;
00167
00168
        met t *met0. *met1:
00169
00170
        gsl_rng *rng[NTHREADS];
00171
00172
        FILE *dirlist;
00173
00174
        char dirname[LEN], filename[LEN];
00175
00176
        double *dt, t, t0;
00177
00178
        int i, ip, ntask = 0, rank = 0, size = 1;
00179
00180 #ifdef MPI
00181
        /* Initialize MPI... */
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
00191
         /* Open directory list... */
        if (!(dirlist = fopen(argv[1], "r")))
00192
00193
          ERRMSG("Cannot open directory list!");
00194
        /* Loop over directories... */
while (fscanf(dirlist, "%s", dirname) != EOF) {
00195
00196
00197
00198
           /* MPI parallelization... */
          if ((++ntask) % size != rank)
00199
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);
ALLOC(met1, met_t, 1);
00211
00212
00213
00214
          ALLOC(dt, double,
```

```
00215
                 NP);
00216
00217
           /\star Read control parameters... \star/
00218
           {\tt sprintf(filename, "\$s/\$s", dirname, argv[2]);}
00219
           read_ctl(filename, argc, argv, &ctl);
00220
           /* Initialize random number generators... */
00222
           gsl_rng_env_setup();
00223
           for (i = 0; i < NTHREADS; i++)
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
           /\star Get simulation time interval... \star/
00231
           init_simtime(&ctl, atm);
00232
           /* 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... */
           STOP_TIMER(TIMER_INIT);
00240
00241
00242
00243
             Loop over timesteps...
00244
00245
00246
           /* Loop over timesteps... */
00247
           for (t = t0; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;</pre>
00248
                t += ctl.direction * ctl.dt_mod) {
00249
             /\star Adjust length of final time step... \star/
00250
00251
             if (ctl.direction * (t - ctl.t_stop) > 0)
               t = ctl.t_stop;
00253
00254
             /\star Set time steps for air parcels... \star/
             for (ip = 0; ip < atm->np; ip++)
  if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
00255
00256
                    && ctl.direction * (atm->time[ip] - ctl.t_stop) <= 0
&& ctl.direction * (atm->time[ip] - t) < 0))
00257
00258
00259
                 dt[ip] = t - atm->time[ip];
00260
               else
00261
                 dt[ip] = GSL_NAN;
00262
00263
             /* Get meteorological data... */
             START_TIMER(TIMER_INPUT);
00264
00265
             get_met(&ctl, argv[4], t, met0, met1);
00266
             STOP_TIMER(TIMER_INPUT);
00267
00268
             /* Initialize isosurface... */
00269
             START_TIMER(TIMER_ISOSURF);
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
{\tt 00276~\#pragma~omp~parallel~for~default(shared)~private(ip)}
            for (ip = 0; ip < atm->np; ip++)
00278
              if (gsl_finite(dt[ip]))
00279
                 module_advection(met0, met1, atm, ip, dt[ip]);
00280
             STOP_TIMER(TIMER_ADVECT);
00281
00282
             /* Turbulent diffusion... */
             START_TIMER(TIMER_DIFFTURB);
00283
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],
00296
00297
                                          rng[omp_get_thread_num()]);
00298
             STOP_TIMER(TIMER_DIFFMESO);
00299
             /* Sedimentation... */
00300
00301
             START_TIMER (TIMER_SEDI);
```

```
00302 #pragma omp parallel for default(shared) private(ip)
         for (ip = 0; ip < atm->np; ip++)
             if (gsl_finite(dt[ip]))
00304
00305
                module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
00306
            STOP_TIMER(TIMER_SEDI);
00307
             /* Isosurface... */
00309
            START_TIMER(TIMER_ISOSURF);
00310 #pragma omp parallel for default(shared) private(ip)
           for (ip = 0; ip < atm->np; ip++)
  module_isosurf(&ctl, met0, met1, atm, ip);
00311
00312
00313
            STOP_TIMER(TIMER_ISOSURF);
00314
00315
             /* Position...
00316
            START_TIMER(TIMER_POSITION);
00317 \#pragma omp parallel for default(shared) private(ip)
            for (ip = 0; ip < atm->np; ip++)
  module_position(met0, met1, atm, ip);
00318
00319
             STOP_TIMER(TIMER_POSITION);
00321
00322
             /* Meteorological data... */
00323
            START_TIMER(TIMER_METEO);
00324 \#pragma omp parallel for default(shared) private(ip)
            for (ip = 0; ip < atm->np; ip++)
  module_meteo(&ctl, met0, met1, atm, ip);
00325
00326
             STOP_TIMER(TIMER_METEO);
00327
00328
00329
00330
            START_TIMER (TIMER_DECAY);
00331 #pragma omp parallel for default(shared) private(ip)
            for (ip = 0; ip < atm->np; ip++)
00332
00333
             if (gsl_finite(dt[ip]))
00334
                 module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00335
            STOP_TIMER(TIMER_DECAY);
00336
             /* Write output... */
00337
00338
            START_TIMER (TIMER_OUTPUT);
             write_output(dirname, &ctl, met0, met1, atm, t);
00340
            STOP_TIMER(TIMER_OUTPUT);
00341
00342
00343
00344
             Finalize model run...
00345
00346
00347
          /* Report timers...
00348
          STOP_TIMER(TIMER_TOTAL);
00349
          PRINT_TIMER (TIMER_TOTAL);
          PRINT_TIMER(TIMER_INIT);
PRINT_TIMER(TIMER_INPUT);
00350
00351
          PRINT_TIMER (TIMER_OUTPUT);
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);
          PRINT_TIMER (TIMER_METEO);
00359
          PRINT_TIMER (TIMER_POSITION);
00360
          PRINT_TIMER (TIMER_SEDI);
00361
00362
          /* Report memory usage... */
          00363
00364
00365
00366
          00367
00368
00369
                   + (2 * GX * GY * GZ) * sizeof(int)) / 1024. / 1024.);
00370
00371
          /* 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());
00372
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
```

5.32 trac.c 163

Here is the call graph for this function:



5.32 trac.c

```
00002
          This file is part of MPTRAC.
00003
          MPTRAC is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by
00004
00005
          the Free Software Foundation, either version 3 of the License, or
00006
00007
          (at your option) any later version.
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 {\tt GmbH}
00018 */
00019
00025 #include "libtrac.h"
```

```
00026
00027 #ifdef MPI
00028 #include "mpi.h"
00029 #endif
00030
00031 /* -
00032 Defines...
00033
00034
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
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(
       met_t * met0,
met_t * met1,
00082
00083
00084
       atm_t * atm,
00085
        int ip,
00086
       double dt);
00087
00089 void module_decay(
       ctl_t * ctl,
00090
00091
        met_t * met0,
00092
        met_t * met1,
00093
        atm_t * atm,
00094
        int ip,
00095
       double dt);
00096
00098 void module_diffusion_meso(
00099
       ctl_t * ctl,
        met_t * met0,
00100
00101
        met_t * met1,
        atm_t * atm,
00102
       int ip,
double dt,
00103
00104
00105
        gsl_rng * rng);
00106
00108 void module_diffusion_turb(
00109
       ctl_t * ctl,
atm_t * atm,
00110
       int ip,
00111
00112
       double dt,
00113
        gsl_rng * rng);
00114
00116 void module_isosurf(
       ctl_t * ctl,
met_t * met0,
00117
00118
00119
       met_t * met1,
00120
       atm_t * atm,
00121
       int ip);
00122
00124 void module meteo(
       ctl_t * ctl,
met_t * met0,
00125
00126
00127
        met_t * met1,
00128
        atm_t * atm,
00129
       int ip);
00130
00132 void module position(
```

5.32 trac.c 165

```
00133
        met_t * met0,
        met_t * met1,
atm_t * atm,
00134
00135
00136
        int ip);
00137
00139 void module_sedi(
        ctl_t * ctl,
00140
00141
        met_t * met0,
        met_t * met1,
atm_t * atm,
00142
00143
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
        Main...
00158
00159
00160 int main(
00161
00162
        char *argv[]) {
00163
00164
        ctl_t ctl;
00165
00166
        atm_t *atm;
00167
00168
        met_t *met0, *met1;
00169
        gsl_rng *rng[NTHREADS];
00170
00171
00172
        FILE *dirlist;
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
00181
        /* Initialize MPI... */
        MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
00182
00183
00184
00185 #endif
00186
00187
         /* Check arguments... */
00188
        if (argc < 5)
          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
00206
           /* Set timers... */
           START_TIMER(TIMER_TOTAL);
00207
00208
          START_TIMER(TIMER_INIT);
00209
           /* Allocate... */
00210
00211
           ALLOC(atm, atm_t, 1);
00212
           ALLOC(met0, met_t, 1);
00213
           ALLOC(met1, met_t, 1);
00214
          ALLOC(dt, double,
00215
                 NP);
00216
00217
           /* Read control parameters... */
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
00237
            t0 = ceil(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00238
00239
          /* Set timers... */
          STOP_TIMER(TIMER_INIT);
00240
00241
00242
00243
            Loop over timesteps...
00244
00245
00246
          /* Loop over timesteps... */
          for (t = t0; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;</pre>
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
            /\star Set time steps for air parcels... \star/
00255
            for (ip = 0; ip < atm\rightarrownp; ip++)
00256
              if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
                   00257
00258
                dt[ip] = t - atm->time[ip];
00260
              else
00261
                dt[ip] = GSL_NAN;
00262
00263
            /* Get meteorological data... */
            START TIMER (TIMER INPUT):
00264
00265
            get_met(&ctl, argv[4], t, met0, met1);
00266
            STOP_TIMER(TIMER_INPUT);
00267
00268
            /* Initialize isosurface... */
00269
            START_TIMER (TIMER_ISOSURF);
00270
            if (t == t0)
  module_isosurf(&ctl, met0, met1, atm, -1);
00271
00272
            STOP_TIMER(TIMER_ISOSURF);
00273
00274
            /* 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]))
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)
            for (ip = 0; ip < atm->np; ip++)
00285
00286
                 (gsl_finite(dt[ip]))
00287
                module_diffusion_turb(&ctl, atm, ip, dt[ip],
00288
                                        rng[omp_get_thread_num()]);
            STOP_TIMER(TIMER_DIFFTURB);
00289
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++)
              if (gsl_finite(dt[ip]))
00295
                module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
00296
00297
                                        rng[omp_get_thread_num()]);
00298
            STOP_TIMER(TIMER_DIFFMESO);
00299
             /* Sedimentation...
00300
            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
00308
            /* Isosurface... */
```

5.32 trac.c 167

```
START_TIMER(TIMER_ISOSURF);
00310 #pragma omp parallel for default(shared) private(ip)
           for (ip = 0; ip < atm->np; ip++)
module_isosurf(&ctl, met0, met1, atm, ip);
00311
00312
00313
            STOP_TIMER(TIMER_ISOSURF);
00314
00315
             /* Position...
00316
            START_TIMER(TIMER_POSITION);
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)
            for (ip = 0; ip < atm->np; ip++)
  module_meteo(&ctl, met0, met1, atm, ip);
00325
00326
            STOP_TIMER(TIMER_METEO);
00328
00329
00330
            START_TIMER(TIMER_DECAY);
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
             STOP_TIMER(TIMER_DECAY);
00335
00336
00337
             /* Write output... */
            START_TIMER(TIMER_OUTPUT);
00338
00339
            write_output(dirname, &ctl, met0, met1, atm, t);
00340
            STOP_TIMER(TIMER_OUTPUT);
00341
00342
00343
             Finalize model run...
00344
00345
00347
           /* Report timers...
00348
          STOP_TIMER(TIMER_TOTAL);
00349
          PRINT_TIMER(TIMER_TOTAL);
00350
          PRINT_TIMER(TIMER_INIT);
          PRINT TIMER (TIMER_INPUT);
00351
00352
          PRINT_TIMER (TIMER_OUTPUT);
          PRINT_TIMER (TIMER_ADVECT);
00353
00354
          PRINT_TIMER(TIMER_DECAY);
00355
          PRINT_TIMER (TIMER_DIFFMESO);
00356
          PRINT_TIMER (TIMER_DIFFTURB);
          PRINT_TIMER(TIMER_ISOSURF);
PRINT_TIMER(TIMER_METEO);
00357
00358
          PRINT_TIMER (TIMER_POSITION);
00360
          PRINT_TIMER(TIMER_SEDI);
00361
          00362
00363
00364
00365
00366
          00367
00368
00369
00370
00371
00372
          /* Report problem size... */
00373
          printf("SIZE_NP = d\n", atm->np);
          printf("SIZE_TASKS = %d\n", size);
printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00374
00375
00376
00377
          /* Free random number generators... */
          for (i = 0; i < NTHREADS; i++)</pre>
00378
00379
            gsl_rng_free(rng[i]);
00380
          /* Free... */
00381
00382
          free(atm);
00383
          free (met0);
00384
          free (met1);
00385
          free(dt);
00386
00387
00388 #ifdef MPT
00389 /* Finalize MPI... */
00390
        MPI_Finalize();
00391 #endif
00392
00393
        return EXIT_SUCCESS;
00394 }
00395
```

```
00397
00398 void init_simtime(
       ctl_t * ctl,
atm_t * atm) {
00399
00400
00401
00402
       /\star Set inital and final time... \star/
00403
       if (ctl->direction == 1)
00404
        if (ctl->t_start < -1e99)</pre>
         ctl->t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
if (ctl->t_stop < -le99)</pre>
00405
00406
          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
00410
           ctl->t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00411
         if (ctl->t_start < -1e99)</pre>
00412
           ctl->t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00413
       }
00414
00415
        /* Check time... */
00416
       if (ctl->direction * (ctl->t_stop - ctl->t_start) <= 0)</pre>
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,
00427
       double dt) {
00428
00429
       double v[3], xm[3];
00430
00431
        /* Interpolate meteorological data... */
       00432
00433
00434
                       &v[0], &v[1], &v[2], NULL, NULL);
00435
       /\star Get position of the mid point... \star/
00436
       xm[0] = atm->lon[ip] + dx2deg(0.5 * dt * v[0] / 1000., atm->lat[ip]);
xm[1] = atm->lat[ip] + dy2deg(0.5 * dt * v[1] / 1000.);
00437
00438
00439
       xm[2] = atm - p[ip] + 0.5 * dt * v[2];
00440
00441
        /* Interpolate meteorological data for mid point...
00442
       intpol_met_time(met0, met1, atm->time[ip] + 0.5 * dt,
                       xm[2], xm[0], xm[1], NULL, NULL,
00443
00444
                       &v[0], &v[1], &v[2], NULL, NULL);
00445
00446
       /* Save new position... */
00447
       atm->time[ip] += dt;
       atm->lon[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 }
00452
00454
00455 void module_decay(
00456
       ctl_t * ctl,
met_t * met0,
00457
00458
       met_t * met1,
00459
       atm_t * atm,
00460
       int ip,
00461
       double dt) {
00462
00463
       double ps, pt, tdec;
00464
00465
       /* Check lifetime values... */
      if ((ctl->tdec_trop <= 0 && ctl->tdec_strat <= 0) || ctl->
00466
     qnt_m < 0
00467
         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... */
```

5.32 trac.c 169

```
pt = tropopause(atm->time[ip], atm->lat[ip]);
00483
            /* Set lifetime... */
00484
           if (atm->p[ip] \ll pt)
00485
00486
             tdec = ctl->tdec strat;
00487
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 }
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,
         double dt,
00503
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]);
           iy = locate(met0->lat, met0->ny, atm->lat[ip]);
00515
           iz = locate(met0->p, met0->np, atm->p[ip]);
00516
00517
           /* Collect local wind data... */
00519
           u[0] = met0 \rightarrow u[ix][iy][iz];
00520
           u[1] = met0 -> u[ix + 1][iy][iz];
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];

u[5] = met0->u[ix + 1][iy][iz + 1];
00522
00523
00524
           u[6] = met0 -> u[ix][iy + 1][iz + 1];
00525
00526
           u[7] = met0 -> u[ix + 1][iy + 1][iz + 1];
00527
           v[0] = met0 \rightarrow v[ix][iy][iz];
00528
           v[1] = met0->v[ix + 1][iy][iz];
v[2] = met0->v[ix][iy + 1][iz];
00529
00530
           v[3] = met0 > v[ix][iy | 1][iz];
v[3] = met0 -> v[ix + 1][iy + 1][iz];
v[4] = met0 -> v[ix][iy][iz + 1];
00532
00533
           v[5] = met0 -> v[ix + 1][iy][iz + 1];
00534
           v[6] = met0 -> v[ix][iy + 1][iz + 1];
           v[7] = met0 -> v[ix + 1][iy + 1][iz + 1];
00535
00536
           w[0] = met0->w[ix][iy][iz];
           w[1] = met0 -> w[ix + 1][iy][iz];
00538
00539
           w[2] = met0->w[ix][iy + 1][iz];
00540
           w[3] = met0 -> w[ix + 1][iy + 1][iz];
           w[4] = met0->w[ix][iy][iz + 1];
w[5] = met0->w[ix + 1][iy][iz + 1];
w[6] = met0->w[ix + 1][iy][iz + 1];
00541
00542
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]);
00551
            /* Collect local wind data... */
00552
           u[8] = met1->u[ix][iy][iz];
           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];
00554
00555
00556
00557
           u[13] = met1 -> u[ix + 1][iy][iz + 1];
00558
           u[14] = met1->u[ix][iy + 1][iz + 1];
00559
           u[15] = met1 -> u[ix + 1][iy + 1][iz + 1];
00560
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];
v[13] = met1->v[ix + 1][iy][iz + 1];
00564
00565
00566
           v[14] = met1 -> v[ix][iy + 1][iz + 1];
00567
```

```
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];
00572
          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
          /\star Get standard deviations of local wind data... \star/
00579
          usig = gsl_stats_sd(u, 1, 16);
vsig = gsl_stats_sd(v, 1, 16);
00580
00581
00582
          wsig = gsl_stats_sd(w, 1, 16);
00583
          /* Set temporal correlations for mesoscale fluctuations... */ r = 1 - 2 * fabs(dt) / ctl->dt_met;
00584
00585
          rs = sqrt(1 - r * r);
00586
00587
00588
          /* Calculate mesoscale wind fluctuations... */
00589
          atm->up[ip] =
00590
           r * atm->up[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
                                                                ctl->turb_meso * usig);
00591
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 * wsia);
00598
00599
          /* Calculate air parcel displacement... */
00600
          atm \rightarrow lon[ip] += dx2deg(atm \rightarrow up[ip] * dt / 1000., atm \rightarrow lat[ip]);
          atm->lat[ip] += dy2deg(atm->vp[ip] * dt / 1000.);
00601
00602
          atm->p[ip] += atm->wp[ip] * dt;
00603
00604 }
00605
00607
00608 void module_diffusion_turb(
00609
       ctl_t * ctl,
atm_t * atm,
00610
00611
        int ip,
        double dt,
00612
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
        if (atm->p[ip] > p0)
00623
00624
          w = 1;
00625
        else if (atm->p[ip] < p1)</pre>
00626
          w = 0;
00627
        else
          w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00628
00629
00630
        /* Set diffusivitiy... */
       dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00631
00632
00633
00634
        /* Horizontal turbulent diffusion... */
00635
        if (dx > 0)  {
         atm->lon[ip]
00636
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.
```

5.32 trac.c 171

```
met_t * met0,
        met_t * met1,
atm_t * atm,
00656
00657
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
        char line[LEN];
00666
00667
        /* Check control parameter... */ if (ctl->isosurf < 1 || ctl->isosurf > 4)
00668
00669
          return;
00670
00671
00672
        /* Initialize... */
00673
        if (ip < 0) {</pre>
00674
00675
           /* Allocate... */
00676
          ALLOC(iso, double,
00677
                NP);
00678
          ALLOC(ps, double,
00679
                 nb);
          ALLOC(ts, double,
00680
                nb);
00681
00682
00683
          /* Save pressure... */
00684
          if (ctl->isosurf == 1)
            for (ip2 = 0; ip2 < atm->np; ip2++)
  iso[ip2] = atm->p[ip2];
00685
00686
00687
          /* Save density... */
00688
00689
          else if (ctl->isosurf == 2)
             for (ip2 = 0; ip2 < atm->np; ip2++) {
00690
              00691
00692
00693
                                NULL, NULL, NULL);
00694
              iso[ip2] = atm->p[ip2] / t;
00695
00696
          /\star Save potential temperature... \star/
00697
00698
          else if (ctl->isosurf == 3)
00699
             for (ip2 = 0; ip2 < atm->np; ip2++) {
00700
               intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00701
                                atm->lon[ip2], atm->lat[ip2], NULL, &t, NULL, NULL,
              NULL, NULL);
iso[ip2] = t * pow(P0 / atm->p[ip2], 0.286);
00702
00703
00704
00705
00706
           /* Read balloon pressure data... */
00707
          else if (ctl->isosurf == 4) {
00708
00709
             /* Write info... */
00710
            printf("Read balloon pressure data: %s\n", ctl->balloon);
00711
00712
             /* Open file... */
            if (!(in = fopen(ctl->balloon, "r")))
    ERRMSG("Cannot open file!");
00713
00714
00715
00716
             /* Read pressure time series... */
            while (fgets(line, LEN, in))
if (sscanf(line, "%lg %lg", &ts[n], &ps[n]) == 2)
00718
00719
                 if ((++n) > 100000)
                   ERRMSG("Too many data points!");
00720
00721
00722
             /\star Check number of points... \star/
            if (n < 1)
00723
00724
              ERRMSG("Could not read any data!");
00725
00726
             /* Close file... */
00727
            fclose(in);
00728
00729
00730
          /* Leave initialization... */
00731
          return;
00732
00733
00734
        /* Restore pressure... */
00735
        if (ctl->isosurf == 1)
          atm \rightarrow p[ip] = iso[ip];
00737
        /* Restore density... */
00738
        else if (ctl->isosurf == 2) {
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... */
else if (ctl->isosurf == 3) {
00746
00747
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
          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
        /\star Interpolate pressure... \star/
00753
        else if (ctl->isosurf == 4) {
00754
         if (atm->time[ip] <= ts[0])</pre>
          atm->p[ip] = ps[0];
else if (atm->time[ip] >= ts[n - 1])
00755
00756
00757
           atm->p[ip] = ps[n - 1];
          else {
00758
           idx = locate(ts, n, atm->time[ip]);
00759
            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,
00772
        atm_t * atm,
00773
        int ip) {
00774
00775
        /* Interpolate surface pressure... */
00776
        if (ctl->qnt_ps >= 0)
         intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00777
     lon[ip],
00778
                           atm->lat[ip], &atm->q[ctl->qnt_ps][ip], NULL,
00779
                          NULL, NULL, NULL, NULL, NULL);
00780
00781
        /* Interpolate temperature... */
00782
        if (ctl->qnt_t >= 0)
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00783
     lon[ip],
00784
                           atm->lat[ip], NULL, &atm->q[ctl->qnt_t][ip],
00785
                          NULL, NULL, NULL, NULL, NULL);
00786
00787
        /* Interpolate zonal wind... */
00788
        if (ctl->qnt_u >= 0)
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00789
     lon[ip],
00790
                           atm->lat[ip], NULL, NULL, &atm->q[ctl->qnt_u][ip],
00791
                          NULL, NULL, NULL, NULL);
00792
00793
        /* Interpolate meridional wind... */
00794
        if (ctl->qnt_v >= 0)
00795
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
     lon[ip],
00796
                          atm->lat[ip], NULL, NULL, NULL,
00797
                          &atm->q[ctl->qnt_v][ip], NULL, NULL, NULL);
00798
00799
        /* Interpolate vertical velocity... */
00800
        if (ctl->qnt_w >= 0)
00801
         intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
                          atm->lat[ip], NULL, NULL, NULL, NULL,
00802
                          &atm->q[ctl->qnt_w][ip], NULL, NULL);
00803
00804
00805
        /* Interpolate water vapor vmr... */
00806
        if (ctl->qnt_h2o >= 0)
00807
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
                          atm->lat[ip], NULL, NULL, NULL, NULL, NULL,
80800
00809
                          &atm->q[ctl->qnt_h2o][ip], NULL);
00810
00811
        /* Interpolate ozone... */
00812
        if (ctl->qnt_o3 >= 0)
00813
          intpol met time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00814
                          atm->lat[ip], NULL, NULL, NULL, NULL, NULL, NULL,
00815
                           &atm->q[ctl->qnt_o3][ip]);
00816
00817
        /* Calculate potential temperature... */
00818
        if (ctl->qnt_theta >= 0) {
00819
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
```

5.32 trac.c 173

```
lon[ip],
00820
                           atm->lat[ip], NULL, &atm->q[ctl->qnt_theta][ip],
00821
                           NULL, NULL, NULL, NULL, NULL);
          atm->q[ctl->qnt\_theta][ip] *= pow(P0 / atm->p[ip], 0.286);
00822
00823
00824 }
00827
00828 void module_position(
00829
        met_t * met0,
        met_t * met1,
atm_t * atm,
00830
00831
00832
        int ip) {
00833
00834
        double ps;
00835
00836
        /* Calculate modulo... */
        atm \rightarrow lon[ip] = fmod(atm \rightarrow lon[ip], 360);
00837
        atm->lat[ip] = fmod(atm->lat[ip], 360);
00838
00839
        /* Check latitude... */
while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
   if (atm->lat[ip] > 90) {
     atm->lat[ip] = 180 - atm->lat[ip];
     atm->lon[ip] += 180;
}
00840
00841
00842
00843
00844
00845
          if (atm->lat[ip] < -90) {
  atm->lat[ip] = -180 - atm->lat[ip];
  atm->lon[ip] += 180;
00846
00847
00848
00849
00850
        }
00851
00852
        /\star Check longitude... \star/
00853
        while (atm->lon[ip] < -180)
        atm->lon[ip] += 360;
while (atm->lon[ip] >= 180)
00854
00855
00856
         atm->lon[ip] -= 360;
00857
00858
        /* Get surface pressure... */
        intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00859
00860
                         atm->lon[ip], atm->lat[ip], &ps, NULL,
00861
                         NULL, NULL, NULL, NULL, NULL);
00862
00863
        /* Check pressure... */
00864
        if (atm->p[ip] > ps)
00865
         atm->p[ip] = ps;
        else if (atm->p[ip] < met0->p[met0->np - 1])
00866
          atm->p[ip] = met0->p[met0->np - 1];
00867
00868 }
00869
00871
00872 void module_sedi(
00873
        ctl_t * ctl,
met_t * met0,
00874
        met_t * met1,
00876
        atm_t * atm,
00877
        int ip,
00878
        double dt) {
00879
00880
        /\star Coefficients for Cunningham slip-flow correction (Kasten, 1968): \star/
00881
        const double A = 1.249, B = 0.42, C = 0.87;
00882
00883
        /\star Specific gas constant for dry air [J/(kg K)]: \star/
00884
        const double R = 287.058;
00885
00886
        /* Average mass of an air molecule [kg/molec]: */
00887
        const double m = 4.8096e-26;
00888
00889
        double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
00890
00891
        /* Check if parameters are available... */
00892
        if (ctl->qnt_r < 0 || ctl->qnt_rho < 0)
00893
         return;
00894
00895
        /* Convert units...
00896
        p = 100 * atm->p[ip];
00897
        r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
        rho_p = atm->q[ctl->qnt_rho][ip];
00898
00899
00900
        /* Get temperature... */
        intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00901
      lon[ip],
00902
                         atm->lat[ip], NULL, &T, NULL, NULL, NULL, NULL, NULL);
00903
00904
        /* Density of dry air... */
```

```
00905
        rho = p / (R * T);
00906
00907
        /* Dynamic viscosity of air... */
        eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
00908
00909
00910
         /* Thermal velocity of an air molecule... */
        v = sqrt(8 * GSL_CONST_MKSA_BOLTZMANN * T / (M_PI * m));
00911
00912
00913
         /* Mean free path of an air molecule... */
00914
        lambda = 2 * eta / (rho * v);
00915
00916
         /* Knudsen number for air... */
00917
        K = lambda / r p;
00918
00919
         /* Cunningham slip-flow correction... */
00920
        G = 1 + K * (A + B * exp(-C / K));
00921
00922
         /* Sedimentation (fall) velocity... */
00923
        v_p =
00924
          2. * gsl_pow_2(r_p) * (rho_p -
00925
                                    rho) * GSL_CONST_MKSA_GRAV_ACCEL / (9. * eta) * G;
00926
        /* Calculate pressure change... */
atm->p[ip] += dz2dp(v_p * dt / 1000., atm->p[ip]);
00927
00928
00929 }
00930
00932
00933 void write_output(
00934
        const char *dirname,
00935
        ctl t * ctl.
00936
        met_t * met0,
00937
00938
        atm_t * atm,
00939
        double t) {
00940
00941
        char filename[LEN];
00942
00943
        double r;
00944
00945
        int year, mon, day, hour, min, sec;
00946
        /* Get time... */
00947
00948
        jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
00949
        /* Write atmospheric data... */
if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
    sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_tab",
00950
00951
00952
                   dirname, ctl->atm_basename, year, mon, day, hour, min);
00953
00954
          write atm(filename, ctl, atm, t);
00955
00956
00957
        /∗ Write gridded data...
        /* white graded value... */
if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
    sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d_tab",
00958
00959
          dirname, ctl->grid_basename, year, mon, day, hour, min); write_grid(filename, ctl, met0, met1, atm, t);
00960
00961
00962
00963
00964
        /* Write CSI data... */
        if (ctl->csi_basename[0] != '-') {
00965
         sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
write_csi(filename, ctl, atm, t);
00966
00967
00968
00969
        00970
00971
00972
00973
          write_prof(filename, ctl, met0, met1, atm, t);
00974
00975
        /* Write station data... */
00976
        /* white station data... */
if (ctl->stat_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
00977
00978
00979
          write_station(filename, ctl, atm, t);
00980
00981 }
```

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
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, 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
            /\star Read control parameters... \star/
00063
            read_ctl(argv[1], argc, argv, &ctl);
           read_ctl(argv[1], argc, argv, &ctl);
t0 = scan_ctl(argv[1], argc, argv, "WIND_TO", -1, "0", NULL);
nx = (int) scan_ctl(argv[1], argc, argv, "WIND_NX", -1, "360", NULL);
ny = (int) scan_ctl(argv[1], argc, argv, "WIND_NY", -1, "181", NULL);
nz = (int) scan_ctl(argv[1], argc, argv, "WIND_NZ", -1, "61", NULL);
z0 = scan_ctl(argv[1], argc, argv, "WIND_ZO", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "WIND_Z1", -1, "60", NULL);
u0 = scan_ctl(argv[1], argc, argv, "WIND_UO", -1, "38.587660177302", NULL);
u1 = scan_ctl(argv[1], argc, argv, "WIND_U1", -1, "38.587660177302", NULL);
alpha = scan_ctl(argv[1], argc, argv, "WIND_LD1", -1, "0.0", NULL);
00064
00065
00066
00067
00068
00069
00070
00071
00072
           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 <= NX <= MAX!");
00077
            if (ny < 1 || ny > EY)
ERRMSG("Set 1 <= NY <= MAX!");</pre>
00078
00079
            if (nz < 1 || nz > EP)
08000
               ERRMSG("Set 1 <= NZ <= MAX!");</pre>
```

```
00081
00082
             /* Get time... */
00083
             jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
            t0 = year * 10000. + mon * 100. + day + hour / 24.;
00084
00085
            /* Set filename... */
sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", argv[2], year, mon, day, hour);
00086
00088
             /* Create netCDF file... */
00089
00090
            NC(nc_create(filename, NC_CLOBBER, &ncid));
00091
00092
             /* Create dimensions... */
            /* Create dimensions... */
NC(nc_def_dim(ncid, "time", 1, &dims[0]));
NC(nc_def_dim(ncid, "lev", (size_t) nz, &dims[1]));
NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[2]));
NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[3]));
00093
00094
00095
00096
00097
00098
            /* 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));
00099
00100
             NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[2], &latid));
00101
            NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[3], &lonid));
00102
            NC(nc_def_var(ncid, "ION", NC_DOUBLE, I, &dims[0], &tid));
NC(nc_def_var(ncid, "U", NC_FLOAT, 4, &dims[0], &tid));
NC(nc_def_var(ncid, "V", NC_FLOAT, 4, &dims[0], &vid));
NC(nc_def_var(ncid, "V", NC_FLOAT, 4, &dims[0], &vid));
NC(nc_def_var(ncid, "W", NC_FLOAT, 4, &dims[0], &wid));
00103
00104
00105
00106
00107
             /* Set attributes...
00108
            /* 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", "long_name", "air_pressure");
add_text_attribute(ncid, "T", "long_name", "Temperature");
add_text_attribute(ncid, "T", "units", "K");
add_text_attribute(ncid, "U", "long_name", "U velocity");
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
00119
            add_text_attribute(ncid, "U", "long_name", "U velocity");
            add_text_attribute(ncid, "U", "units", "m s**-1");
add_text_attribute(ncid, "V", "long_name", "V velocity");
00120
00121
            add_text_attribute(ncid, "V", "units", "m s**-1");
add_text_attribute(ncid, "W", "long_name", "Vertical velocity");
add_text_attribute(ncid, "W", "units", "Pa s**-1");
00122
00123
00124
00125
             /* End definition... */
00126
00127
            NC(nc_enddef(ncid));
00128
00129
             /* Set coordinates... */
00130
            for (ix = 0; ix < nx; ix++)
               dataLon[ix] = 360.0 / nx * (double) ix;
00131
            for (iy = 0; iy < ny; iy++)
dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
00132
00133
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)... */
            for (ix = 0; ix < nx; ix++)
00144
                for (iy = 0; iy < ny; iy++)</pre>
00145
00146
                   for (iz = 0; iz < nz; iz++) {</pre>
00147
                      idx = (iz * ny + iy) * nx + ix;

dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
00148
                                                           * (cos(dataLat[iy] * M_PI / 180.0)

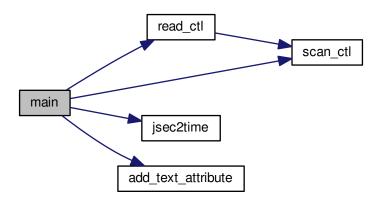
* cos(alpha * M_PI / 180.0)
00149
00150
                                                                + sin(dataLat[iy] * M_PI / 180.0)

* cos(dataLon[ix] * M_PI / 180.0)
00151
00152
                      * sin(alpha * M_PI / 180.0)));
dataV[idx] = (float) (-LIN(0.0, u0, nz - 1.0, u1, iz)
* sin(dataLon[ix] * M_PI / 180.0)
00153
00154
00155
                                                           * sin(alpha * M_PI / 180.0));
00156
00157
00158
             /* Write wind data... */
00159
            NC(nc_put_var_float(ncid, tid, dataT));
00160
            NC(nc_put_var_float(ncid, uid, dataU));
00161
00162
             NC(nc_put_var_float(ncid, vid, dataV));
            NC(nc_put_var_float(ncid, wid, dataW));
00163
00164
00165
             /* Close file... */
00166
            NC (nc_close (ncid));
00167
```

5.34 wind.c 177

```
00168    return EXIT_SUCCESS;
00169 }
```

Here is the call graph for this function:



5.34 wind.c

```
00001 /*
00002
         This file is part of MPTRAC.
00003
00004
         MPTRAC is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by
00005
00006
         the Free Software Foundation, either version 3 of the License, or
00007
         (at your option) any later version.
00008
00009
         \ensuremath{\mathsf{MPTRAC}} is distributed in the hope that it will be useful,
00010
         but WITHOUT ANY WARRANTY; without even the implied warranty of
00011
         MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
         GNU General Public License for more details.
00013
         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 /* -----
00028
          Functions...
00029
00030
00031 void add_text_attribute(
00032
         int ncid,
         char *varname,
char *attrname,
00033
00034
00035
         char *text);
00036
00037 /* --
00038
00039
00040
00041 int main(
00042
         int argc,
00043
         char *argv[]) {
00044
00045
         ctl_t ctl;
00046
00047
         static char filename[LEN]:
00048
         static double r, t0, z0, z1, dataLon[EX], dataLat[EY], dataZ[EP],
```

```
u0, u1, alpha;
00050
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, vid, wid,
00056
                idx, ix, iy, iz, nx, ny, nz, year, mon, day, hour, min, sec;
00057
00058
              /* Check arguments... */
00059
             if (argc < 3)
                ERRMSG("Give parameters: <ctl> <metbase>");
00060
00061
00062
             /* Read control parameters... */
             read_ctl(argv[1], argc, argv, &ctl);
00063
            t0 = scan_ctl(argv[1], argc, argv, "WIND_TO", -1, "0", NULL);

nx = (int) scan_ctl(argv[1], argc, argv, "WIND_NX", -1, "360", NULL);

ny = (int) scan_ctl(argv[1], argc, argv, "WIND_NY", -1, "181", NULL);

nz = (int) scan_ctl(argv[1], argc, argv, "WIND_NZ", -1, "61", NULL);

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

z1 = scan_ctl(argv[1], argc, argv, "WIND_Z1", -1, "60", NULL);

u0 = scan_ctl(argv[1], argc, argv, "WIND_UO", -1, "38.587660177302", NULL);

u1 = scan_ctl(argv[1], argc, argv, "WIND_U1", -1, "38.587660177302", NULL);

alpha = scan_ctl(argv[1], argc, argv, "WIND_L1PHA". -1. "0 0" NULL);
00064
             t0 = scan_ctl(argv[1], argc, argv, "WIND_TO", -1, "0", NULL);
00065
00066
00067
00069
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>
00077
             if (ny < 1 || ny > EY)
               ERRMSG("Set 1 <= NY <= MAX!");
00078
00079
             if (nz < 1 || nz > EP)
00080
               ERRMSG("Set 1 <= NZ <= MAX!");
00081
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
00088
00089
              /* Create netCDF file... */
00090
             NC(nc_create(filename, NC_CLOBBER, &ncid));
00091
00092
              /* Create dimensions... */
            /* Create dimensions... */
NC(nc_def_dim(ncid, "time", 1, &dims[0]));
NC(nc_def_dim(ncid, "lev", (size_t) nz, &dims[1]));
NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[2]));
NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[3]));
00093
00094
00095
00096
00097
00098
              /* 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));
00099
00100
             NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[2], &latid));
00101
             NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[3], &lonid));
00102
             NC(nc_def_var(ncid, "Ion", NC_DOUBLE, 1, &dims[3], &lon NC(nc_def_var(ncid, "T", NC_FLOAT, 4, &dims[0], &tid)); NC(nc_def_var(ncid, "U", NC_FLOAT, 4, &dims[0], &uid)); NC(nc_def_var(ncid, "V", NC_FLOAT, 4, &dims[0], &vid)); NC(nc_def_var(ncid, "W", NC_FLOAT, 4, &dims[0], &wid));
00103
00104
00105
00107
00108
            /* 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");
              /* Set attributes... */
00109
00110
00111
00112
00113
00114
00115
00116
00117
             add_text_attribute(ncid, "T", "units", "K");
add_text_attribute(ncid, "U", "long_name", "U velocity");
00118
00119
00120
             add_text_attribute(ncid, "U", "units", "m s**-1");
             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", "units", "Pa s**-1");
00122
00123
00124
00125
00126
              /* End definition... */
00127
             NC(nc_enddef(ncid));
00128
00129
              /* Set coordinates... */
00130
             for (ix = 0; ix < nx; ix++)
                dataLon[ix] = 360.0 / nx * (double) ix;
00131
              for (iy = 0; iy < ny; iy++) dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
00132
00133
00134
             for (iz = 0; iz < nz; iz++)
                dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));
00135
00136
```

5.34 wind.c 179

```
/* 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++)
00146
                                for (iz = 0; iz < nz; iz++) {</pre>
                                     or (1z = 0; 1z \ 12; 1z \ 1; 1; 1z \ 1
00147
00148
00149
00150
                                    + sin(dataLat[iy] * M_PI / 180.0)

* cos(dataLon[ix] * M_PI / 180.0)

* sin(alpha * M_PI / 180.0)));

dataV[idx] = (float) (-LIN(0.0, u0, nz - 1.0, u1, iz)

* sin(dataLon[ix] * M_PI / 180.0)
00151
00152
00153
00154
00155
00156
                                                                                                  * sin(alpha * M_PI / 180.0));
00157
00158
                     /\star Write wind data... \star/
00159
                    NC(nc_put_var_float(ncid, tid, dataT));
NC(nc_put_var_float(ncid, uid, dataU));
00160
00161
00162
                     NC(nc_put_var_float(ncid, vid, dataV));
00163
                     NC(nc_put_var_float(ncid, wid, dataW));
00164
00165
                      /* Close file... */
                    NC(nc_close(ncid));
00166
00167
00168
                     return EXIT_SUCCESS;
00169 }
00170
00172
00173 void add_text_attribute(
00174 int ncid,
00175
                    char *varname,
00176
                    char *attrname,
00177
                    char *text) {
00178
00179
                    int varid:
00180
                     NC(nc_inq_varid(ncid, varname, &varid));
00182
                     NC(nc_put_att_text(ncid, varid, attrname, strlen(text), text));
00183 }
```

Index

ctl_t, 5

index	
add text attribute	atm_basename, 11
wind.c, 175	atm_dt_out, 12
atm basename	atm_gpfile, 12
ctl_t, 11	balloon, 11
atm dt out	csi basename, 12
ctl t, 12	_
atm_gpfile	csi_dt_out, 12
ct_t, 12	csi_lat0, 13
atm_t, 3	csi_lat1, 13
lat, 4	csi_lon0, 13
lon, 4	csi_lon1, 13
	csi_modmin, 12
np, 4 p, 4	csi_nx, 13
φ, 4 q, 4	csi_ny, 13
time, 4	csi_nz, 12
up, 4	csi_obsfile, 12
υρ, 4 vp, 4	csi_obsmin, 12
wp, 5	csi_z0, 12
Ψρ , 3	csi_z1, 13
balloon	direction, 10
ctl_t, 11	dt_met, 10
- '	dt_mod, 10
cart2geo	grid_basename, 13
libtrac.c, 39	grid_dt_out, 14
libtrac.h, 87	grid_gpfile, 13
center.c, 19	grid_lat0, 15
main, 20	grid_lat1, 15
csi_basename	grid_lon0, 14 grid_lon1, 14
ctl_t, 12	grid_nx, 14
csi_dt_out	grid_ny, 14
ctl_t, 12	grid_nz, 14
csi_lat0	grid_sparse, 14
ctl_t, 13	grid_sparse, 14 grid_z0, 14
csi_lat1	grid_z1, 14
ctl_t, 13	isosurf, 10
csi_lon0	nq, 8
ctl_t, 13	prof_basename, 15
csi_lon1	prof lat0, 16
ctl_t, 13	prof_lat1, 16
csi_modmin ctl t, 12	prof lon0, 15
	prof lon1, 16
csi_nx	prof_nx, 15
ctl_t, 13	prof_ny, 16
csi_ny ctl_t, 13	prof_nz, 15
csi_nz	prof_obsfile, 15
ctl_t, 12	prof_z0, 15
csi_obsfile	prof_z1, 15
ctl_t, 12	qnt_format, 8
csi_obsmin	qnt_h2o, 9
ctl_t, 12	qnt_m, 9
csi_z0	qnt_name, 8
ctl_t, 12	qnt_o3, 10
csi_z1	qnt_ps, 9
ctl t, 13	qnt_r, 9
otl t 5	ant rho 9

qnt_rho, 9

qnt_stat, 10	libtrac.h, 90
qnt_t, 9	grid_basename
qnt_theta, 10	ctl_t, 13
qnt_u, 9	grid_dt_out
qnt_unit, 8	ctl_t, 14
qnt_v, 9	grid_gpfile
qnt_w, 9	ctl_t, 13
stat_basename, 16 stat_lat, 16	grid_lat0
stat_lon, 16	ctl_t, 15 grid_lat1
stat_r, 16	ctl_t, 15
t_start, 10	grid_lon0
t_stop, 10	ctl_t, 14
tdec_strat, 11	grid_lon1
tdec_trop, 11	ctl_t, 14
turb_dx_strat, 11	grid_nx
turb_dx_trop, 11	ctl_t, 14
turb_dz_strat, 11	grid_ny
turb_dz_trop, 11	ctl_t, 14
turb_meso, 11	grid_nz
de vO de	ctl_t, 14
deg2dx	grid_sparse
libtrac.c, 39 libtrac.h, 87	ctl_t, 14
deg2dy	grid_z0
libtrac.c, 39	ctl_t, 14
libtrac.h, 87	grid_z1 ctl_t, 14
direction	Cti_t, 14
ctl_t, 10	h2o
dist.c, 24	met_t, 19
main, 24	
, = .	
dp2dz	init.c, 32
dp2dz libtrac.c, 39	main, 33
dp2dz libtrac.c, 39 libtrac.h, 88	main, 33 init_simtime
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met	main, 33 init_simtime trac.c, 150
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10	main, 33 init_simtime trac.c, 150 intpol_met_2d
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90 intpol_met_3d
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90 intpol_met_3d libtrac.c, 42
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 88	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90 intpol_met_3d libtrac.c, 42 libtrac.h, 91
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 88 dy2deg	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90 intpol_met_3d libtrac.c, 42 libtrac.h, 91 intpol_met_space
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 88 dy2deg libtrac.c, 40	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90 intpol_met_3d libtrac.c, 42 libtrac.h, 91
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 88 dy2deg	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90 intpol_met_3d libtrac.c, 42 libtrac.h, 91 intpol_met_space libtrac.c, 43
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 88 dy2deg libtrac.c, 40 libtrac.h, 88	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90 intpol_met_3d libtrac.c, 42 libtrac.h, 91 intpol_met_space libtrac.c, 43 libtrac.h, 91
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 88 dy2deg libtrac.c, 40 libtrac.h, 88 dy2deg	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90 intpol_met_3d libtrac.c, 42 libtrac.h, 91 intpol_met_space libtrac.c, 43 libtrac.h, 91 intpol_met_time
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 88 dy2deg libtrac.c, 40 libtrac.h, 88 dz2dp libtrac.c, 40 libtrac.h, 88	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90 intpol_met_3d libtrac.c, 42 libtrac.h, 91 intpol_met_space libtrac.c, 43 libtrac.h, 91 intpol_met_time libtrac.c, 44
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 88 dy2deg libtrac.c, 40 libtrac.h, 88 dz2dp libtrac.c, 40 libtrac.h, 88 dz2dp libtrac.c, 40 libtrac.h, 88 extract.c, 30	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90 intpol_met_3d libtrac.c, 42 libtrac.h, 91 intpol_met_space libtrac.c, 43 libtrac.h, 91 intpol_met_time libtrac.c, 44 libtrac.h, 92
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 88 dy2deg libtrac.c, 40 libtrac.h, 88 dz2dp libtrac.c, 40 libtrac.h, 88	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90 intpol_met_3d libtrac.c, 42 libtrac.h, 91 intpol_met_space libtrac.c, 43 libtrac.h, 91 intpol_met_time libtrac.c, 44 libtrac.h, 92 isosurf ctl_t, 10
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 88 dy2deg libtrac.c, 40 libtrac.h, 88 dz2dp libtrac.c, 40 libtrac.h, 88 dz2dp extract.c, 40 libtrac.h, 88	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90 intpol_met_3d libtrac.c, 42 libtrac.h, 91 intpol_met_space libtrac.c, 43 libtrac.h, 91 intpol_met_time libtrac.c, 44 libtrac.h, 92 isosurf ctl_t, 10
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 88 dy2deg libtrac.c, 40 libtrac.h, 88 dz2dp libtrac.c, 40 libtrac.h, 88 dz2dp libtrac.c, 40 libtrac.h, 88 extract.c, 30 main, 30 geo2cart	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90 intpol_met_3d libtrac.c, 42 libtrac.h, 91 intpol_met_space libtrac.c, 43 libtrac.h, 91 intpol_met_time libtrac.c, 44 libtrac.h, 92 isosurf ctl_t, 10 jsec2time libtrac.c, 45
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 88 dy2deg libtrac.c, 40 libtrac.h, 88 dz2dp libtrac.c, 40 libtrac.h, 88 extract.c, 40 libtrac.h, 88 extract.c, 30 main, 30 geo2cart libtrac.c, 40	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90 intpol_met_3d libtrac.c, 42 libtrac.h, 91 intpol_met_space libtrac.c, 43 libtrac.h, 91 intpol_met_time libtrac.c, 44 libtrac.h, 92 isosurf ctl_t, 10 jsec2time libtrac.c, 45 libtrac.h, 93
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 88 dy2deg libtrac.c, 40 libtrac.h, 88 dz2dp libtrac.c, 40 libtrac.h, 88 extract.c, 30 main, 30 geo2cart libtrac.h, 88	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90 intpol_met_3d libtrac.c, 42 libtrac.h, 91 intpol_met_space libtrac.c, 43 libtrac.h, 91 intpol_met_time libtrac.c, 44 libtrac.h, 92 isosurf ctl_t, 10 jsec2time libtrac.c, 45 libtrac.h, 93 jsec2time.c, 36
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 88 dy2deg libtrac.c, 40 libtrac.h, 88 dz2dp libtrac.c, 40 libtrac.h, 88 extract.c, 30 main, 30 geo2cart libtrac.h, 88 get_met	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90 intpol_met_3d libtrac.c, 42 libtrac.h, 91 intpol_met_space libtrac.c, 43 libtrac.h, 91 intpol_met_time libtrac.c, 44 libtrac.h, 92 isosurf ctl_t, 10 jsec2time libtrac.c, 45 libtrac.h, 93
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 88 dy2deg libtrac.c, 40 libtrac.h, 88 dz2dp libtrac.c, 40 libtrac.h, 88 extract.c, 30 main, 30 geo2cart libtrac.c, 40 libtrac.h, 88 get_met libtrac.c, 40	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90 intpol_met_3d libtrac.c, 42 libtrac.h, 91 intpol_met_space libtrac.c, 43 libtrac.h, 91 intpol_met_time libtrac.c, 44 libtrac.h, 92 isosurf ctl_t, 10 jsec2time libtrac.c, 45 libtrac.h, 93 jsec2time.c, 36
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 88 dy2deg libtrac.c, 40 libtrac.h, 88 dz2dp libtrac.c, 40 libtrac.h, 88 extract.c, 30 main, 30 geo2cart libtrac.h, 88 get_met	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90 intpol_met_3d libtrac.c, 42 libtrac.h, 91 intpol_met_space libtrac.c, 43 libtrac.h, 91 intpol_met_time libtrac.c, 44 libtrac.h, 92 isosurf ctl_t, 10 jsec2time libtrac.h, 93 jsec2time.c, 36 main, 36
dp2dz libtrac.c, 39 libtrac.h, 88 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 88 dy2deg libtrac.c, 40 libtrac.h, 88 dz2dp libtrac.c, 40 libtrac.h, 88 extract.c, 30 main, 30 geo2cart libtrac.c, 40 libtrac.h, 88 get_met libtrac.c, 40 libtrac.h, 89	main, 33 init_simtime trac.c, 150 intpol_met_2d libtrac.c, 42 libtrac.h, 90 intpol_met_3d libtrac.c, 42 libtrac.h, 91 intpol_met_space libtrac.c, 43 libtrac.h, 91 intpol_met_time libtrac.c, 44 libtrac.h, 92 isosurf ctl_t, 10 jsec2time libtrac.h, 93 jsec2time.c, 36 main, 36

libtrac.c, 37	tropopause, 102
cart2geo, 39	write_atm, 104
deg2dx, 39	write_csi, 105
deg2dy, 39	write_grid, 107
dp2dz, 39	write_prof, 109
dx2deg, 40	write_station, 111
dy2deg, 40	locate
dz2dp, 40	libtrac.c, 45
geo2cart, 40	libtrac.h, 93
get_met, 40	lon
get met help, 41	atm_t, 4
intpol_met_2d, 42	met_t, 18
intpol_met_3d, 42	
intpol_met_space, 43	main
intpol_met_time, 44	center.c, 20
jsec2time, 45	dist.c, 24
locate, 45	extract.c, 30
read_atm, 46	init.c, 33
read_ctl, 46	jsec2time.c, 36
	match.c, 119
read_met, 49	met_map.c, 124
read_met_extrapolate, 51	met_prof.c, 128
read_met_help, 51	met_sample.c, 133
read_met_periodic, 52	met_zm.c, 136
scan_ctl, 52	smago.c, 140
time2jsec, 53	split.c, 144
timer, 53	time2jsec.c, 148
tropopause, 54	trac.c, 160
write_atm, 56	wind.c, 175
write_csi, 57	match.c, 119
write_grid, 59	main, 119
write_prof, 61	met_map.c, 124
write_station, 63	main, 124
libtrac.h, 85	met_prof.c, 128
cart2geo, 87	main, 128
deg2dx, 87	met_sample.c, 132
deg2dy, 87	main, 133
dp2dz, 88	met_t, 17
dx2deg, 88	h2o, 19
dy2deg, 88	lat, 18
dz2dp, 88	lon, 18
geo2cart, 88	np, 18
get_met, 89	nx, 18
get_met_help, 90	ny, 18
intpol_met_2d, 90	o3, 19
intpol_met_3d, 91	p, 18
intpol_met_space, 91	ps, 18
intpol_met_time, 92	t, 19
jsec2time, 93	time, 18
locate, 93	u, 19
read_atm, 94	v, 19
read_ctl, 94	w, 19
read_met, 97	met_zm.c, 135
read_met_extrapolate, 99	main, 136
read_met_help, 99	module_advection
read_met_periodic, 100	trac.c, 150
scan_ctl, 100	module_decay
time2jsec, 101	trac.c, 150
timer, 101	module_diffusion_meso

trac.c, 151	qnt_m
module_diffusion_turb	ctl_t, 9
trac.c, 153	qnt_name
module_isosurf	ctl_t, 8
trac.c, 154	qnt_o3
module_meteo	ctl_t, 10
trac.c, 156	qnt_ps
module position	ctl t, 9
trac.c, 157	qnt_r
module_sedi	ctl_t, 9
trac.c, 158	qnt_rho
	ctl_t, 9
np	qnt_stat
atm_t, 4	ctl_t, 10
met_t, 18	qnt_t
nq	ctl_t, 9
ctl_t, 8	qnt_theta
nx	ctl_t, 10
met_t, 18	qnt_u
ny	ctl_t, 9
met t, 18	gnt unit
- /	ctl_t, 8
03	qnt_v
met t, 19	ctl_t, 9
р	qnt_w
atm_t, 4	ctl_t, 9
met_t, 18	road atm
	read_atm
prof_basename	libtrac.c, 46
ctl_t, 15	libtrac.h, 94
prof_lat0	read_ctl
ctl_t, 16	libtrac.c, 46
prof_lat1	libtrac.h, 94
ctl_t, 16	read_met
prof_lon0	libtrac.c, 49
ctl_t, 15	libtrac.h, 97
prof_lon1	read_met_extrapolate
ctl_t, 16	libtrac.c, 51
and an	
prof_nx	libtrac.h, 99
pror_nx ctl_t, 15	libtrac.h, 99 read met help
ctl_t, 15	read_met_help
ctl_t, 15 prof_ny	read_met_help libtrac.c, 51
ctl_t, 15 prof_ny ctl_t, 16	read_met_help libtrac.c, 51 libtrac.h, 99
ctl_t, 15 prof_ny ctl_t, 16 prof_nz	read_met_help libtrac.c, 51 libtrac.h, 99 read_met_periodic
ctl_t, 15 prof_ny ctl_t, 16 prof_nz ctl_t, 15	read_met_help libtrac.c, 51 libtrac.h, 99 read_met_periodic libtrac.c, 52
ctl_t, 15 prof_ny ctl_t, 16 prof_nz ctl_t, 15 prof_obsfile	read_met_help libtrac.c, 51 libtrac.h, 99 read_met_periodic
ctl_t, 15 prof_ny ctl_t, 16 prof_nz ctl_t, 15 prof_obsfile ctl_t, 15	read_met_help libtrac.c, 51 libtrac.h, 99 read_met_periodic libtrac.c, 52 libtrac.h, 100
ctl_t, 15 prof_ny ctl_t, 16 prof_nz ctl_t, 15 prof_obsfile ctl_t, 15 prof_z0	read_met_help libtrac.c, 51 libtrac.h, 99 read_met_periodic libtrac.c, 52 libtrac.h, 100 scan_ctl
ctl_t, 15 prof_ny ctl_t, 16 prof_nz ctl_t, 15 prof_obsfile ctl_t, 15 prof_z0 ctl_t, 15	read_met_help libtrac.c, 51 libtrac.h, 99 read_met_periodic libtrac.c, 52 libtrac.h, 100 scan_ctl libtrac.c, 52
ctl_t, 15 prof_ny ctl_t, 16 prof_nz ctl_t, 15 prof_obsfile ctl_t, 15 prof_z0 ctl_t, 15 prof_z1	read_met_help libtrac.c, 51 libtrac.h, 99 read_met_periodic libtrac.c, 52 libtrac.h, 100 scan_ctl libtrac.c, 52 libtrac.h, 100
ctl_t, 15 prof_ny ctl_t, 16 prof_nz ctl_t, 15 prof_obsfile ctl_t, 15 prof_z0 ctl_t, 15 prof_z1 ctl_t, 15	read_met_help libtrac.c, 51 libtrac.h, 99 read_met_periodic libtrac.c, 52 libtrac.h, 100 scan_ctl libtrac.c, 52 libtrac.h, 100 smago.c, 140
ctl_t, 15 prof_ny ctl_t, 16 prof_nz ctl_t, 15 prof_obsfile ctl_t, 15 prof_z0 ctl_t, 15 prof_z1 ctl_t, 15 ps	read_met_help libtrac.c, 51 libtrac.h, 99 read_met_periodic libtrac.c, 52 libtrac.h, 100 scan_ctl libtrac.c, 52 libtrac.h, 100 smago.c, 140 main, 140
ctl_t, 15 prof_ny ctl_t, 16 prof_nz ctl_t, 15 prof_obsfile ctl_t, 15 prof_z0 ctl_t, 15 prof_z1 ctl_t, 15	read_met_help libtrac.c, 51 libtrac.h, 99 read_met_periodic libtrac.c, 52 libtrac.h, 100 scan_ctl libtrac.c, 52 libtrac.h, 100 smago.c, 140 main, 140 split.c, 143
ctl_t, 15 prof_ny ctl_t, 16 prof_nz ctl_t, 15 prof_obsfile ctl_t, 15 prof_z0 ctl_t, 15 prof_z1 ctl_t, 15 ps met_t, 18	read_met_help libtrac.c, 51 libtrac.h, 99 read_met_periodic libtrac.c, 52 libtrac.h, 100 scan_ctl libtrac.c, 52 libtrac.h, 100 smago.c, 140 main, 140 split.c, 143 main, 144
ctl_t, 15 prof_ny ctl_t, 16 prof_nz ctl_t, 15 prof_obsfile ctl_t, 15 prof_z0 ctl_t, 15 prof_z1 ctl_t, 15 ps	read_met_help libtrac.c, 51 libtrac.h, 99 read_met_periodic libtrac.c, 52 libtrac.h, 100 scan_ctl libtrac.c, 52 libtrac.h, 100 smago.c, 140 main, 140 split.c, 143 main, 144 stat_basename
ctl_t, 15 prof_ny ctl_t, 16 prof_nz ctl_t, 15 prof_obsfile ctl_t, 15 prof_z0 ctl_t, 15 prof_z1 ctl_t, 15 ps met_t, 18	read_met_help libtrac.c, 51 libtrac.h, 99 read_met_periodic libtrac.c, 52 libtrac.h, 100 scan_ctl libtrac.c, 52 libtrac.h, 100 smago.c, 140 main, 140 split.c, 143 main, 144
ctl_t, 15 prof_ny ctl_t, 16 prof_nz ctl_t, 15 prof_obsfile ctl_t, 15 prof_z0 ctl_t, 15 prof_z1 ctl_t, 15 ps met_t, 18	read_met_help libtrac.c, 51 libtrac.h, 99 read_met_periodic libtrac.c, 52 libtrac.h, 100 scan_ctl libtrac.c, 52 libtrac.h, 100 smago.c, 140 main, 140 split.c, 143 main, 144 stat_basename
ctl_t, 15 prof_ny ctl_t, 16 prof_nz ctl_t, 15 prof_obsfile ctl_t, 15 prof_z0 ctl_t, 15 prof_z1 ctl_t, 15 ps met_t, 18 q atm_t, 4	read_met_help libtrac.c, 51 libtrac.h, 99 read_met_periodic libtrac.c, 52 libtrac.h, 100 scan_ctl libtrac.c, 52 libtrac.h, 100 smago.c, 140 main, 140 split.c, 143 main, 144 stat_basename ctl_t, 16
ctl_t, 15 prof_ny ctl_t, 16 prof_nz ctl_t, 15 prof_obsfile ctl_t, 15 prof_z0 ctl_t, 15 prof_z1 ctl_t, 15 ps met_t, 18 q atm_t, 4 qnt_format	read_met_help libtrac.c, 51 libtrac.h, 99 read_met_periodic libtrac.c, 52 libtrac.h, 100 scan_ctl libtrac.c, 52 libtrac.h, 100 smago.c, 140 main, 140 split.c, 143 main, 144 stat_basename ctl_t, 16 stat_lat
ctl_t, 15 prof_ny ctl_t, 16 prof_nz ctl_t, 15 prof_obsfile ctl_t, 15 prof_z0 ctl_t, 15 prof_z1 ctl_t, 15 ps met_t, 18 q atm_t, 4 qnt_format ctl_t, 8	read_met_help libtrac.c, 51 libtrac.h, 99 read_met_periodic libtrac.c, 52 libtrac.h, 100 scan_ctl libtrac.c, 52 libtrac.h, 100 smago.c, 140 main, 140 split.c, 143 main, 144 stat_basename ctl_t, 16 stat_lat ctl_t, 16

stat r	W
 ctl_t, 16	met_t, 19
	wind.c, 174
t	add_text_attribute, 175
met_t, 19	main, 175
t start	wp
ctl_t, 10	atm_t, 5
t_stop	write_atm
ctl_t, 10	libtrac.c, 56
tdec_strat	libtrac.h, 104
 ctl_t, 11	write_csi
tdec_trop	libtrac.c, 57
 ctl_t, 11	libtrac.h, 105
time	write_grid
atm_t, 4	libtrac.c, 59
met_t, 18	libtrac.h, 107
time2jsec	write_output
libtrac.c, 53	trac.c, 159
libtrac.h, 101	write_prof
time2jsec.c, 147	libtrac.c, 61
main, 148	libtrac.h, 109
timer	write_station
libtrac.c, 53	libtrac.c, 63
libtrac.h, 101	libtrac.h, 111
trac.c, 149	,
init_simtime, 150	
main, 160	
module_advection, 150	
module_decay, 150	
module_diffusion_meso, 151	
module_diffusion_turb, 153	
module_isosurf, 154	
module_meteo, 156	
module_position, 157	
module_sedi, 158	
write_output, 159	
tropopause	
libtrac.c, 54	
libtrac.h, 102	
turb_dx_strat	
ctl_t, 11	
turb_dx_trop	
ctl_t, 11	
turb_dz_strat	
ctl_t, 11	
turb_dz_trop	
ctl_t, 11	
turb_meso	
ctl_t, 11	
u	
met_t, 19	
up	
atm_t, 4	
V	
met_t, 19	
vp	
atm_t, 4	