# **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	20
	5.1	center.	c File Reference	. 20
		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	66
5.13	libtrac.h File Reference	87
	5.13.1 Detailed Description	89
	5.13.2 Function Documentation	89
5.14	libtrac.h	116
5.15	match.c File Reference	122
	5.15.1 Detailed Description	122
	5.15.2 Function Documentation	123
5.16	match.c	125
5.17	met_map.c File Reference	127
	5.17.1 Detailed Description	127
	5.17.2 Function Documentation	127
5.18	met_map.c	129
5.19	met prof.c File Reference	131
		131
		131
5.20	met prof.c	
	met_sample.c File Reference	

	5.21.1 Detailed Description	135
	5.21.2 Function Documentation	136
5.22	met_sample.c	137
5.23	met_zm.c File Reference	138
	5.23.1 Detailed Description	138
	5.23.2 Function Documentation	139
5.24	met_zm.c	141
5.25	smago.c File Reference	143
	5.25.1 Detailed Description	143
	5.25.2 Function Documentation	143
5.26	smago.c	145
5.27	split.c File Reference	146
	5.27.1 Detailed Description	146
	5.27.2 Function Documentation	147
5.28	split.c	149
5.29	time2jsec.c File Reference	150
	5.29.1 Detailed Description	150
	5.29.2 Function Documentation	151
5.30	time2jsec.c	151
5.31	trac.c File Reference	152
	5.31.1 Detailed Description	152
	5.31.2 Function Documentation	153
5.32	trac.c	166
5.33	wind.c File Reference	177
	5.33.1 Detailed Description	178
	5.33.2 Function Documentation	178
5.34	wind.c	180

183

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	20
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	87
match.c Calculate deviations between two trajectories	122

127
131
135
138
143
146
150
152
177

Vertical velocity perturbation [hPa/s].

4

```
4.1.1 Detailed Description
Atmospheric data.
Definition at line 410 of file libtrac.h.
4.1.2 Field Documentation
4.1.2.1 int atm_t::np
Number of air pacels.
Definition at line 413 of file libtrac.h.
4.1.2.2 double atm_t::time[NP]
Time [s].
Definition at line 416 of file libtrac.h.
4.1.2.3 double atm_t::p[NP]
Pressure [hPa].
Definition at line 419 of file libtrac.h.
4.1.2.4 double atm_t::lon[NP]
Longitude [deg].
Definition at line 422 of file libtrac.h.
4.1.2.5 double atm_t::lat[NP]
Latitude [deg].
Definition at line 425 of file libtrac.h.
4.1.2.6 double atm_t::q[NQ][NP]
Quantitiy data (for various, user-defined attributes).
Definition at line 428 of file libtrac.h.
4.1.2.7 double atm_t::up[NP]
Zonal wind perturbation [m/s].
Definition at line 431 of file libtrac.h.
```

```
4.1.2.8 double atm_t::vp[NP]
Meridional wind perturbation [m/s].
Definition at line 434 of file libtrac.h.
4.1.2.9 double atm_t::wp[NP]
Vertical velocity perturbation [hPa/s].
Definition at line 437 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 met\_np

Number of target pressure levels.

double met\_p [EP]

Target pressure levels [hPa].

· int isosurf

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

• char balloon [LEN]

Balloon position filename.

double turb\_dx\_trop

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

double turb dx strat

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

double turb\_dz\_trop

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

• double turb\_dz\_strat

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

· 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<sup>^</sup>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::ng 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::met_np
Number of target pressure levels.
Definition at line 239 of file libtrac.h.
4.2.2.23 double ctl_t::met_p[EP]
Target pressure levels [hPa].
Definition at line 242 of file libtrac.h.
4.2.2.24 int ctl_t::isosurf
Isosurface parameter (0=none, 1=pressure, 2=density, 3=theta, 4=balloon).
Definition at line 246 of file libtrac.h.
4.2.2.25 char ctl_t::balloon[LEN]
Balloon position filename.
Definition at line 249 of file libtrac.h.
4.2.2.26 double ctl_t::turb_dx_trop
Horizontal turbulent diffusion coefficient (troposphere) [m^2/s].
Definition at line 252 of file libtrac.h.
4.2.2.27 double ctl_t::turb_dx_strat
Horizontal turbulent diffusion coefficient (stratosphere) [m^2/s].
Definition at line 255 of file libtrac.h.
4.2.2.28 double ctl_t::turb_dz_trop
Vertical turbulent diffusion coefficient (troposphere) [m<sup>2</sup>/s].
Definition at line 258 of file libtrac.h.
4.2.2.29 double ctl_t::turb_dz_strat
Vertical turbulent diffusion coefficient (stratosphere) [m<sup>2</sup>/s].
Definition at line 261 of file libtrac.h.
```

```
4.2.2.30 double ctl_t::turb_meso
Scaling factor for mesoscale wind fluctuations.
Definition at line 264 of file libtrac.h.
4.2.2.31 double ctl_t::tdec_trop
Life time of particles (troposphere) [s].
Definition at line 267 of file libtrac.h.
4.2.2.32 double ctl_t::tdec_strat
Life time of particles (stratosphere) [s].
Definition at line 270 of file libtrac.h.
4.2.2.33 char ctl_t::atm_basename[LEN]
Basename of atmospheric data files.
Definition at line 273 of file libtrac.h.
4.2.2.34 char ctl_t::atm_gpfile[LEN]
Gnuplot file for atmospheric data.
Definition at line 276 of file libtrac.h.
4.2.2.35 double ctl_t::atm_dt_out
Time step for atmospheric data output [s].
Definition at line 279 of file libtrac.h.
4.2.2.36 char ctl_t::csi_basename[LEN]
Basename of CSI data files.
Definition at line 282 of file libtrac.h.
4.2.2.37 double ctl_t::csi_dt_out
Time step for CSI data output [s].
Definition at line 285 of file libtrac.h.
4.2.2.38 char ctl_t::csi_obsfile[LEN]
Observation data file for CSI analysis.
Definition at line 288 of file libtrac.h.
```

4.2.2.39 double ctl\_t::csi\_obsmin Minimum observation index to trigger detection. Definition at line 291 of file libtrac.h. 4.2.2.40 double ctl\_t::csi\_modmin Minimum column density to trigger detection [kg/m<sup>2</sup>]. Definition at line 294 of file libtrac.h. 4.2.2.41 int ctl\_t::csi\_nz Number of altitudes of gridded CSI data. Definition at line 297 of file libtrac.h. 4.2.2.42 double ctl\_t::csi\_z0 Lower altitude of gridded CSI data [km]. Definition at line 300 of file libtrac.h. 4.2.2.43 double ctl\_t::csi\_z1 Upper altitude of gridded CSI data [km]. Definition at line 303 of file libtrac.h. 4.2.2.44 int ctl\_t::csi\_nx Number of longitudes of gridded CSI data. Definition at line 306 of file libtrac.h. 4.2.2.45 double ctl\_t::csi\_lon0 Lower longitude of gridded CSI data [deg]. Definition at line 309 of file libtrac.h. 4.2.2.46 double ctl\_t::csi\_lon1 Upper longitude of gridded CSI data [deg]. Definition at line 312 of file libtrac.h. 4.2.2.47 int ctl\_t::csi\_ny Number of latitudes of gridded CSI data.

Definition at line 315 of file libtrac.h.

```
4.2.2.48 double ctl_t::csi_lat0
Lower latitude of gridded CSI data [deg].
Definition at line 318 of file libtrac.h.
4.2.2.49 double ctl_t::csi_lat1
Upper latitude of gridded CSI data [deg].
Definition at line 321 of file libtrac.h.
4.2.2.50 char ctl_t::grid_basename[LEN]
Basename of grid data files.
Definition at line 324 of file libtrac.h.
4.2.2.51 char ctl_t::grid_gpfile[LEN]
Gnuplot file for gridded data.
Definition at line 327 of file libtrac.h.
4.2.2.52 double ctl_t::grid_dt_out
Time step for gridded data output [s].
Definition at line 330 of file libtrac.h.
4.2.2.53 int ctl_t::grid_sparse
Sparse output in grid data files (0=no, 1=yes).
Definition at line 333 of file libtrac.h.
4.2.2.54 int ctl_t::grid_nz
Number of altitudes of gridded data.
Definition at line 336 of file libtrac.h.
4.2.2.55 double ctl_t::grid_z0
Lower altitude of gridded data [km].
Definition at line 339 of file libtrac.h.
4.2.2.56 double ctl_t::grid_z1
Upper altitude of gridded data [km].
Definition at line 342 of file libtrac.h.
```

```
4.2.2.57 int ctl_t::grid_nx
Number of longitudes of gridded data.
Definition at line 345 of file libtrac.h.
4.2.2.58 double ctl_t::grid_lon0
Lower longitude of gridded data [deg].
Definition at line 348 of file libtrac.h.
4.2.2.59 double ctl_t::grid_lon1
Upper longitude of gridded data [deg].
Definition at line 351 of file libtrac.h.
4.2.2.60 int ctl_t::grid_ny
Number of latitudes of gridded data.
Definition at line 354 of file libtrac.h.
4.2.2.61 double ctl_t::grid_lat0
Lower latitude of gridded data [deg].
Definition at line 357 of file libtrac.h.
4.2.2.62 double ctl_t::grid_lat1
Upper latitude of gridded data [deg].
Definition at line 360 of file libtrac.h.
4.2.2.63 char ctl_t::prof_basename[LEN]
Basename for profile output file.
Definition at line 363 of file libtrac.h.
4.2.2.64 char ctl_t::prof_obsfile[LEN]
Observation data file for profile output.
Definition at line 366 of file libtrac.h.
4.2.2.65 int ctl_t::prof_nz
Number of altitudes of gridded profile data.
```

Definition at line 369 of file libtrac.h.

```
4.2.2.66 double ctl_t::prof_z0
Lower altitude of gridded profile data [km].
Definition at line 372 of file libtrac.h.
4.2.2.67 double ctl_t::prof_z1
Upper altitude of gridded profile data [km].
Definition at line 375 of file libtrac.h.
4.2.2.68 int ctl_t::prof_nx
Number of longitudes of gridded profile data.
Definition at line 378 of file libtrac.h.
4.2.2.69 double ctl_t::prof_lon0
Lower longitude of gridded profile data [deg].
Definition at line 381 of file libtrac.h.
4.2.2.70 double ctl_t::prof_lon1
Upper longitude of gridded profile data [deg].
Definition at line 384 of file libtrac.h.
4.2.2.71 int ctl_t::prof_ny
Number of latitudes of gridded profile data.
Definition at line 387 of file libtrac.h.
4.2.2.72 double ctl_t::prof_lat0
Lower latitude of gridded profile data [deg].
Definition at line 390 of file libtrac.h.
4.2.2.73 double ctl_t::prof_lat1
Upper latitude of gridded profile data [deg].
Definition at line 393 of file libtrac.h.
4.2.2.74 char ctl_t::stat_basename[LEN]
Basename of station data file.
Definition at line 396 of file libtrac.h.
```

```
4.2.2.75 double ctl_t::stat_lon
Longitude of station [deg].
Definition at line 399 of file libtrac.h.
4.2.2.76 double ctl_t::stat_lat
Latitude of station [deg].
Definition at line 402 of file libtrac.h.
4.2.2.77 double ctl_t::stat_r
Search radius around station [km].
Definition at line 405 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 pl [EX][EY][EP]
          Pressure on model levels [hPa].
    float t [EX][EY][EP]
           Temperature [K].

 float u [EX][EY][EP]

          Zonal wind [m/s].
    float v [EX][EY][EP]
          Meridional wind [m/s].
    float w [EX][EY][EP]
           Vertical wind [hPa/s].
    float h2o [EX][EY][EP]
          Water vapor volume mixing ratio [1].

    float o3 [EX][EY][EP]

          Ozone volume mixing ratio [1].
```

```
4.3.1 Detailed Description
Meteorological data.
Definition at line 442 of file libtrac.h.
4.3.2 Field Documentation
4.3.2.1 double met_t::time
Time [s].
Definition at line 445 of file libtrac.h.
4.3.2.2 int met_t::nx
Number of longitudes.
Definition at line 448 of file libtrac.h.
4.3.2.3 int met_t::ny
Number of latitudes.
Definition at line 451 of file libtrac.h.
4.3.2.4 int met_t::np
Number of pressure levels.
Definition at line 454 of file libtrac.h.
4.3.2.5 double met_t::lon[EX]
Longitude [deg].
Definition at line 457 of file libtrac.h.
4.3.2.6 double met_t::lat[EY]
Latitude [deg].
Definition at line 460 of file libtrac.h.
4.3.2.7 double met_t::p[EP]
Pressure [hPa].
Definition at line 463 of file libtrac.h.
```

```
4.3.2.8 double met_t::ps[EX][EY]
Surface pressure [hPa].
Definition at line 466 of file libtrac.h.
4.3.2.9 float met_t::pl[EX][EY][EP]
Pressure on model levels [hPa].
Definition at line 469 of file libtrac.h.
4.3.2.10 float met_t::t[EX][EY][EP]
Temperature [K].
Definition at line 472 of file libtrac.h.
4.3.2.11 float met_t::u[EX][EY][EP]
Zonal wind [m/s].
Definition at line 475 of file libtrac.h.
4.3.2.12 float met_t::v[EX][EY][EP]
Meridional wind [m/s].
Definition at line 478 of file libtrac.h.
4.3.2.13 float met_t::w[EX][EY][EP]
Vertical wind [hPa/s].
Definition at line 481 of file libtrac.h.
4.3.2.14 float met_t::h2o[EX][EY][EP]
Water vapor volume mixing ratio [1].
Definition at line 484 of file libtrac.h.
4.3.2.15 float met_t::o3[EX][EY][EP]
Ozone volume mixing ratio [1].
Definition at line 487 of file libtrac.h.
The documentation for this struct was generated from the following file:
```

Generated by Doxygen

· 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

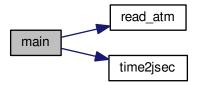
### 5.1.2.1 int main (int argc, char \* argv[])

Definition at line 28 of file center.c.

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

```
"# $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"
00077
                   "# $18 = longitude (90%% percentile) [deg]\n"
00078
00079
                   "# $19 = longitude (maximum) [deg]\n");
         fprintf(out,
00081
                   "# $20 = latitude (mean) [deg] \n
00082
                   "# $21 = latitude (sigma) [deg] n"
                   "# $22 = latitude (minimum) [deg] \n"
00083
                   "# $23 = latitude (10%% percentile) [deg]\n"
00084
                   "# $24 = latitude (1st quarter) [deg] \n"
00085
00086
                   "# $25 = latitude (median) [deg]\n"
                   "# $26 = latitude (3rd quarter) [deg]\n"
00087
00088
                   "# $27 = latitude (90%% percentile) [deg]\n"
                   "# $28 = latitude (maximum) [deg] \n\n");
00089
00090
00091
         /* Loop over files... */
         for (f = 2; f < argc; f++) {
00092
00093
00094
            /* Read atmopheric data... */
00095
           read_atm(argv[f], &ctl, atm);
00096
00097
           /* Initialize... */
00098
           zm = zs = 0;
            lonm = lons = 0;
00099
00100
            latm = lats = 0;
00101
00102
            /\star Calculate mean and standard deviation... \star/
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;
lats += gsl_pow_2(atm->lat[ip]) / atm->np;
00108
00109
00110
00111
00112
            /* Normalize... */
00113
           zs = sqrt(zs - gsl_pow_2(zm));
           lons = sqrt(lons - gsl_pow_2(lonm));
lats = sqrt(lats - gsl_pow_2(latm));
00114
00115
00116
00117
            /* Sort arrays... */
           gsl_sort(atm->p, 1, (size_t) atm->np);
00118
00119
            gsl_sort(atm->lon, 1, (size_t) atm->np);
00120
            gsl_sort(atm->lat, 1, (size_t) atm->np);
00121
00122
            /* Get date from filename... */
           for (i = (int) strlen(argv[f]) - 1; argv[f][i] != '/' || i == 0; i--);
name = strtok(&(argv[f][i]), "_");
00123
00124
           mame = strtok(NULL, "_");
year = strtok(NULL, "_");
mon = strtok(NULL, "_");
day = strtok(NULL, "_");
hour = strtok(NULL, "_");
00125
00126
00127
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
00135
00136
00137
                     Z(atm->p[atm->np - atm->np / 10]),
Z(atm->p[atm->np - atm->np / 4]),
00138
00139
                     Z(atm->p[atm->np - atm->np / 4],
Z(atm->p[atm->np / 2]), Z(atm->p[atm->np / 4]),
Z(atm->p[atm->np / 10]), Z(atm->p[0]),
lonm, lons, atm->lon[0], atm->lon[atm->np / 10],
00140
00141
00142
00143
                     atm->lon[atm->np / 4], atm->lon[atm->np / 2],
                     atm->lon[atm->np - atm->np / 4],
atm->lon[atm->np - atm->np / 10],
00144
00145
                     atm->lon[atm->np - 1],
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],
00148
00150
                     atm->lat[atm->np - atm->np / 10], atm->lat[atm->np - 1]);
00151
00152
         /* Close file... */
00153
00154
        fclose(out);
00155
00156
         /* Free... */
00157
         free(atm);
00158
00159
         return EXIT_SUCCESS;
00160 }
```

Here is the call graph for this function:



### 5.2 center.c

```
00001 /*
00002
         This file is part of MPTRAC.
00003
         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);

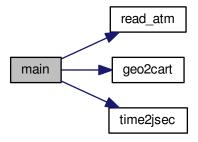
00239 free(dh);

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[])

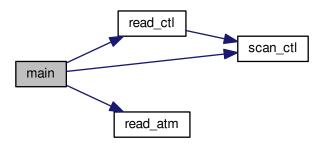
Definition at line 28 of file extract.c.

```
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

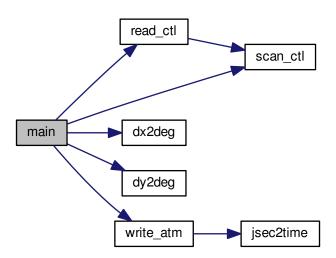
```
5.7.2.1 int main (int argc, char * argv[])
```

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);
dlon = scan_ctl(argv[1], argc, argv, "INIT_LONO1", -1, "0", NULL);
dlon = scan_ctl(argv[1], argc, argv, "INIT_LONO1", -1, "0", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "INIT_LAT0", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT1", -1, "0", NULL);
dlat = scan_ctl(argv[1], argc, argv, "INIT_SLAT1", -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 (ctl\_t \*ctl, char \*filename, met\_t \*met)

Read meteorological data file.

void read met extrapolate (met t \*met)

Extrapolate meteorological data at lower boundary.

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

Read and convert variable from meteorological data file.

void read\_met\_ml2pl (ctl\_t \*ctl, met\_t \*met, float var[EX][EY][EP])

Convert meteorological data from model levels to pressure levels.

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

Create meteorological data with periodic boundary conditions.

 double scan\_ctl (const char \*filename, int argc, char \*argv[], const char \*varname, int arridx, const char \*defvalue, char \*value)

Read a control parameter from file or command line.

void time2jsec (int year, int mon, int day, int hour, int min, int sec, double remain, double \*jsec)

Convert date to seconds.

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

Measure wall-clock time.

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

Write atmospheric data.

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

Write CSI data.

- void write\_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.

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

5.11.2.4 double dp2dz ( double dp, double p )

Convert pressure to vertical distance.

Definition at line 62 of file libtrac.c.

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

5.11.2.5 double dx2deg ( double dx, double lat )

Convert horizontal distance to degrees.

Definition at line 71 of file libtrac.c.

```
00073 {
00074
00075    /* 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 }
```

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.

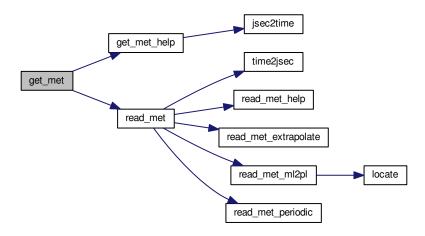
```
5.11.2.9 void get_met ( ctl_t * ctl, char * metbase, double t, met_t * met0, met_t * met1 )
```

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
          read_met(ctl, filename, met0);
00133
00134
00135
          get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
     dt_met, filename);
00136
         read_met(ctl, filename, met1);
00137
00138
00139
        /* Read new data for forward trajectories... */
00140
        if (t > met1->time && ctl->direction == 1) {
00141
        memcpy(met0, met1, sizeof(met_t));
          get_met_help(t, 1, metbase, ctl->dt_met, filename);
read_met(ctl, filename, met1);
00142
00143
00144
00145
00146
        /* Read new data for backward trajectories... */
00147
        if (t < met0->time && ctl->direction == -1) {
00148
        memcpy(met1, met0, sizeof(met_t));
          get_met_help(t, -1, metbase, ctl->dt_met, filename);
read_met(ctl, filename, met0);
00149
00150
00151
00152 }
```

Here is the call graph for this function:



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

Get meteorological data for timestep.

Definition at line 156 of file libtrac.c.

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

Here is the call graph for this function:



5.11.2.11 void intpol\_met\_2d ( double array[EX][EY], int ix, int iy, double wx, double wy, double \* var )

Linear interpolation of 2-D meteorological data.

Definition at line 182 of file libtrac.c.

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

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

5.11.2.12 void intpol\_met\_3d ( float array[EX][EY][EP], int ip, int ix, int iy, double wp, double wx, double wy, double \* var )

Linear interpolation of 3-D meteorological data.

Definition at line 206 of file libtrac.c.

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

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

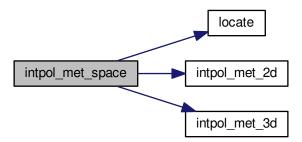
Spatial interpolation of meteorological data.

Definition at line 236 of file libtrac.c.

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

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

Here is the call graph for this function:



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

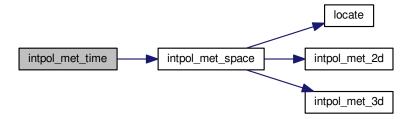
Temporal interpolation of meteorological data.

Definition at line 286 of file libtrac.c.

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

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

Here is the call graph for this function:



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

Convert seconds to date.

Definition at line 341 of file libtrac.c.

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

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

Find array index.

Definition at line 376 of file libtrac.c.

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

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

Read atmospheric data.

Definition at line 408 of file libtrac.c.

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

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

Read control parameters.

Definition at line 458 of file libtrac.c.

```
00462
00463
00464
        int ip, iq;
00465
00466
        /* Write info... */
        printf("\nMassive-Parallel\ Trajectory\ Calculations\ (MPTRAC)\n"
00467
                 "(executable: %s \mid 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);
         ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
00547
00548
         if (ctl->met_np > EP)
           ERRMSG("Too many levels!");
00549
00550
          for (ip = 0; ip < ctl->met_np; ip++)
           ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
00551
00552
00553
          /* Isosurface parameters... */
00554
         ctl->isosurf
         = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00555
00556
          /* Diffusion parameters... */
00558
00559
         ctl->turb dx trop
           = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00560
00561
         ctl->turb_dx_strat
           = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00562
00563
         ctl->turb_dz_trop
00564
            = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00565
         ctl->turb_dz_strat
00566
           = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00567
         ctl->turb meso :
           scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00568
00569
00570
         /* Life time of particles... */
         ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
ctl->tdec_strat =
00571
00572
00573
           scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00574
         /* Output of atmospheric data... */
scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
00575
      atm_basename);
00577
         scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00578
         ctl->atm_dt_out =
           scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00579
00580
00581
         /* Output of CSI data... */
00582
         scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
00583 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",
00584
00585
00586
                   ctl->csi obsfile);
         ctl->csi_obsmin =
00588
           scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00589
         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);
00590
00591
00592
00594
         scan_ctl(filename, argc, argv, "CSI_LONO", -1, "-180", NULL);
ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
00595
00596
00597
         ct1->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);
00598
00599
00600
00601
         ctl->csi_ny =
00602
            (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
00603
00604
         /* Output of grid data... */
         scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
00605
                    ctl->grid_basename);
         scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
00607
       grid_gpfile);
00608
         ctl->grid_dt_out =
           scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
00609
00610
         ctl->grid sparse
            (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
00611
         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);
00612
00613
         ctl->grid nz =
00614
            (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00615
00616
         ctl->grid lon0 =
00617
           scan_ctl(filename, argc, argv, "GRID_LONO", -1, "-180", NULL);
00618
         ctl->grid lon1
00619
           scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
00620
         ctl->grid_nx =
           (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00621
00622
         ctl->grid_lat0 =
```

```
scan_ctl(filename, argc, argv, "GRID_LATO", -1, "-90", NULL);
00624
00625
          scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
00626
        ctl->grid_ny =
           (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
00627
00628
00629
        /* Output of profile data... */
00630
        scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
00631
                   ctl->prof_basename);
00632
        scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
      prof_obsfile);
00633 ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
00634 ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
        ctl->prof_nz =
00635
00636
          (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
00637
        ctl->prof_lon0 =
          scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
00638
00639
        ctl->prof lon1 =
00640
          scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
00641
        ctl->prof_nx =
00642
           (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00643
        ctl->prof_lat0 =
00644
          scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
00645
        ct.1->prof lat.1 =
00646
          scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
00647
        ctl->prof_ny :
00648
          (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00649
00650
        /* Output of station data... */
        00651
00652
        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);
00653
00654
00655
00656 }
```

Here is the call graph for this function:



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

Read meteorological data file.

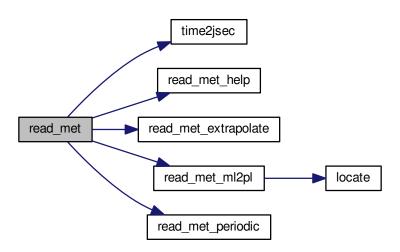
Definition at line 660 of file libtrac.c.

```
00663
00664
       char tstr[10]:
00665
00666
00667
       static float help[EX * EY];
00669
       int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00670
00671
       size_t np, nx, ny;
00672
00673
        /* Write info... */
00674
       printf("Read meteorological data: %s\n", filename);
00675
00676
        /* Get time from filename... */
       sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00677
00678
       year = atoi(tstr);
00679
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00680
       mon = atoi(tstr);
```

```
sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
          day = atoi(tstr);
sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00682
00683
          hour = atoi(tstr);
00684
          time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00685
00686
           /* Open netCDF file... */
00687
00688
          NC(nc_open(filename, NC_NOWRITE, &ncid));
00689
          /* Get dimensions... */
NC(nc_inq_dimid(ncid, "lon", &dimid));
NC(nc_inq_dimlen(ncid, dimid, &nx));
00690
00691
00692
00693
              (nx > EX)
00694
             ERRMSG("Too many longitudes!");
00695
00696
          NC(nc_inq_dimid(ncid, "lat", &dimid));
00697
          NC(nc_inq_dimlen(ncid, dimid, &ny));
00698
          if (ny > EY)
             ERRMSG("Too many latitudes!");
00699
00700
00701
          NC(nc_inq_dimid(ncid, "lev", &dimid));
          NC(nc_inq_dimlen(ncid, dimid, &np));
00702
00703
          if (np > EP)
00704
            ERRMSG("Too many levels!");
00705
00706
          /* Store dimensions... */
          met->np = (int) np;
met->nx = (int) nx;
00707
00708
00709
          met->ny = (int) ny;
00710
00711
          /* Get horizontal grid... */
00712
          NC(nc_inq_varid(ncid, "lon", &varid));
          NC(nc_get_var_double(ncid, varid, met->lon));
NC(nc_inq_varid(ncid, "lat", &varid));
00713
00714
00715
          NC(nc_get_var_double(ncid, varid, met->lat));
00716
00717
          /* Read meteorological data... */
         /* Read meteorological data... */
read_met_help(ncid, "t", "T", met, met->t, 1.0);
read_met_help(ncid, "u", "U", met, met->u, 1.0);
read_met_help(ncid, "v", "V", met, met->v, 1.0);
read_met_help(ncid, "w", "W", met, met->v, 1.0);
read_met_help(ncid, "w", "W", met, met->v, 0.01f);
read_met_help(ncid, "q", "Q", met, met->help(ncid, "o3", "03", met, met->o3, 0.602f);
00718
00719
00720
00721
00722
00723
00724
00725
          /* Meteo data on pressure levels... */
00726
          if (ctl->met_np <= 0) {</pre>
00727
            /* Read pressure levels from file... */
NC(nc_inq_varid(ncid, "lev", &varid));
00728
00729
            NC(nc_get_var_double(ncid, varid, met->p));
for (ip = 0; ip < met->np; ip++)
00730
00731
00732
               met->p[ip] /= 100.;
00733
00734
             /\star Extrapolate data for lower boundary... \star/
00735
            read_met_extrapolate(met);
00736
00737
00738
          /* Meteo data on model levels... */
00739
          else {
00740
00741
             /* Read pressure data from file... */
             read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00742
00743
00744
             /\star Interpolate from model levels to pressure levels... \star/
00745
             read_met_ml2pl(ctl, met, met->t);
00746
             read_met_ml2pl(ctl, met, met->u);
00747
             read_met_ml2pl(ctl, met, met->v);
00748
             read_met_ml2pl(ctl, met, met->w);
00749
             read_met_ml2pl(ctl, met, met->h2o);
00750
             read_met_ml2pl(ctl, met, met->o3);
00751
00752
             /* Set pressure levels... */
            met->np = ctl->met_np;
for (ip = 0; ip < met->np; ip++)
  met->p[ip] = ctl->met_p[ip];
00753
00754
00755
00756
00757
00758
          /* Check ordering of pressure levels... */
          for (ip = 1; ip < met->np; ip++)
  if (met->p[ip - 1] < met->p[ip])
    ERRMSG("Pressure levels must be descending!");
00759
00760
00761
00762
          /* Read surface pressure... */
if (nc_ing_varid(ncid, "PS", &varid) == NC_NOERR) {
00763
00764
00765
            NC(nc_get_var_float(ncid, varid, help));
00766
             for (iy = 0; iy < met->ny; iy++)
for (ix = 0; ix < met->nx; ix++)
00767
```

```
met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
} else if (nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00769
00770
           NC(nc_get_var_float(ncid, varid, help));
           for (iy = 0; iy < met->ny; iy++)
for (ix = 0; ix < met->nx; ix++)
00771
00772
00773
                met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00774
         } else
00775
           for (ix = 0; ix < met->nx; ix++)
00776
             for (iy = 0; iy < met->ny; iy++)
                met \rightarrow ps[ix][iy] = met \rightarrow p[0];
00777
00778
00779
         /* Create periodic boundary conditions... */
00780
         read_met_periodic(met);
00781
00782
         /* Close file... */
00783 NC(nc_close(ncid));
00784 }
```

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 788 of file libtrac.c.

```
00789
00790
00791
         int ip, ip0, ix, iy;
00792
00793
         /* Loop over columns... */
00794
         for (ix = 0; ix < met->nx; ix++)
00795
          for (iy = 0; iy < met->ny; iy++) {
00796
00797
              /* Find lowest valid data point... */
for (ip0 = met->np - 1; ip0 >= 0; ip0--)
   if (!gsl_finite(met->t[ix][iy][ip0])
00798
00800
                     || !gsl_finite(met->u[ix][iy][ip0])
00801
                     || !gsl_finite(met->v[ix][iy][ip0])
00802
                     || !gsl_finite(met->w[ix][iy][ip0]))
00803
                  break;
00804
00805
              /* Extrapolate... */
00806
              for (ip = ip0; ip >= 0; ip--) {
```

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

Read and convert variable from meteorological data file.

Definition at line 819 of file libtrac.c.

```
00826
00827
         static float help[EX * EY * EP];
00828
00829
         int ip, ix, iy, n = 0, varid;
00830
00831
         /* Check if variable exists... */
         if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00832
00833
           if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00834
              return;
00835
         /* Read data... */
NC(nc_get_var_float(ncid, varid, help));
00836
00837
00838
00839
          /* Copy and check data... */
00840
         for (ip = 0; ip < met->np; ip++)
            for (iy = 0; iy < met->ny; iy++)
for (ix = 0; ix < met->nx; ix++) {
  dest[ix][iy][ip] = scl * help[n++];
00841
00842
00843
                 if (fabs(dest[ix][iy][ip] / scl) > 1e14)
  dest[ix][iy][ip] = GSL_NAN;
00845
00846
              }
00847 }
```

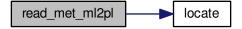
5.11.2.22 void read\_met\_ml2pl ( ctl\_t \* ctl, met\_t \* met, float var[EX][EY][EP] )

Convert meteorological data from model levels to pressure levels.

Definition at line 851 of file libtrac.c.

```
00854
00855
00856
        double aux[EP], p[EP], pt;
00857
00858
        int ip, ip2, ix, iy;
00859
00860
        /* Loop over columns... */
00861
        for (ix = 0; ix < met->nx; ix++)
00862
          for (iy = 0; iy < met->ny; iy++) {
00863
00864
             /* Copy pressure profile... */
for (ip = 0; ip < met->np; ip++)
00865
               p[ip] = met->pl[ix][iy][ip];
00866
00867
00868
             /* Interpolate... */
             for (ip = 0; ip < ctl->met_np; ip++) {
00869
00870
               pt = ctl->met_p[ip];
00871
               if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
                pt = p[0];
00872
00873
               else if ((pt > p[met->np - 1] && p[1] > p[0])
                 || (pt < p[met->np - 1] && p[1] < p[0]))
pt = p[met->np - 1];
00874
00875
               00876
00877
00878
00879
00880
             /* Copy data... */
for (ip = 0; ip < ctl->met_np; ip++)
    var[ix][iy][ip] = (float) aux[ip];
00881
00882
00883
00884
00885 }
```

Here is the call graph for this function:



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

Create meteorological data with periodic boundary conditions.

Definition at line 889 of file libtrac.c.

```
00890
00891
00892
       int ip, iy;
00893
00894
       /* Check longitudes... */
       if (!(fabs(met->lon[met->nx - 1] - met->lon[0])
00895
                  + met->lon[1] - met->lon[0] - 360) < 0.01))
00897
00898
00899
       /\star Increase longitude counter... \star/
       if ((++met->nx) > EX)
00900
00901
         ERRMSG("Cannot create periodic boundary conditions!");
00902
       00903
00904
     lon[0];
00905
00906
        /* Loop over latitudes and pressure levels... */
       for (iy = 0; iy < met->ny; iy++)
00907
        for (ip = 0; ip < met->np; ip++) {
00908
           met->ps[met->nx - 1][iy] = met->ps[0][iy];
met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
00909
00910
00911
           met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
00912
           00913
00915
00916
00917 }
```

5.11.2.24 double scan\_ctl ( const char \* filename, int argc, char \* argv[], const char \* varname, int arridx, const char \* defvalue, char \* value )

Read a control parameter from file or command line.

Definition at line 921 of file libtrac.c.

```
00928
                     {
00929
       FILE *in = NULL:
00930
00931
       char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
00932
00933
        msg[LEN], rvarname[LEN], rval[LEN];
00934
00935
       int contain = 0, i;
00936
00937
       /* Open file... */
00938
       if (filename[strlen(filename) - 1] != '-')
         if (!(in = fopen(filename, "r")))
```

```
00940
               ERRMSG("Cannot open file!");
00941
00942
          /* Set full variable name... */
          if (arridx >= 0) {
00943
           sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
00944
00945
         } else {
          sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
00947
00948
00949
00950
00951
          /* Read data... */
00952
          if (in != NULL)
          while (fgets(line, LEN, in))
if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
if (strcasecmp(rvarname, fullname1) == 0 ||
strcasecmp(rvarname, fullname2) == 0) {
00953
00954
00955
00956
00957
                    contain = 1;
00958
                    break;
00959
                 }
00960
          for (i = 1; i < argc - 1; i++)
00961
          if (strcasecmp(argv[i], fullname1) == 0 ||
               strcasecmp(argv[i], fullname2) == 0) {
sprintf(rval, "%s", argv[i + 1]);
00962
00963
00964
               contain = 1;
00965
               break;
00966
00967
00968
         /* Close file... */
         if (in != NULL)
00969
00970
            fclose(in);
00971
00972
         /* Check for missing variables... */
00973
          if (!contain) {
           if (strlen(defvalue) > 0)
   sprintf(rval, "%s", defvalue);
00974
00975
00976
            else {
00977
              sprintf(msg, "Missing variable %s!\n", fullname1);
00978
               ERRMSG(msg);
00979
00980
00981
00982
         /* Write info... */
         printf("%s = %s\n", fullname1, rval);
00983
00984
00985
          /* Return values... */
         if (value != NULL)
    sprintf(value, "%s", rval);
00986
00987
00988
         return atof(rval);
00989 }
```

5.11.2.25 void time2jsec ( int year, int mon, int day, int hour, int min, int sec, double remain, double \* jsec )

Convert date to seconds.

Definition at line 993 of file libtrac.c.

```
01001
01002
01003
       struct tm t0, t1;
01004
       t0.tm_year = 100;
01005
01006
        t0.tm_mon = 0;
01007
        t0.tm_mday = 1;
01008
        t0.tm\_hour = 0;
       t0.tm_min = 0;
t0.tm_sec = 0;
01009
01010
01011
01012
        t1.tm_year = year - 1900;
        t1.tm_mon = mon - 1;
01014
        t1.tm_mday = day;
01015
        t1.tm_hour = hour;
01016
        t1.tm_min = min;
        t1.tm_sec = sec;
01017
01018
       *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01020 }
```

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

Measure wall-clock time.

Definition at line 1024 of file libtrac.c.

```
01027
01028
01029
       static double starttime[NTIMER], runtime[NTIMER];
01030
        /* Check id...
01031
01032
        if (id < 0 || id >= NTIMER)
01033
         ERRMSG("Too many timers!");
01034
01035
        /* Start timer... */
01036
       if (mode == 1) {
        if (starttime[id] <= 0)</pre>
01037
01038
            starttime[id] = omp_get_wtime();
01039
           ERRMSG("Timer already started!");
01040
01041
01042
01043
       /* Stop timer... */
01044
       else if (mode == 2) {
01045
        if (starttime[id] > 0) {
01046
           runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
            starttime[id] = -1;
01047
01048
         } else
01049
           ERRMSG("Timer not started!");
01050
01051
01052
       /* Print timer... */
01053
       else if (mode == 3)
         printf("%s = %g s\n", name, runtime[id]);
01054
01055 }
```

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

Definition at line 1059 of file libtrac.c.

```
01061
01062
           static double doys[12]
01063
           = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01064
01065
01066
            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,
01067
01068
               -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,
01069
01070
01072
                45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
01073
               75, 77.5, 80, 82.5, 85, 87.5, 90
01074
           };
01075
01076
           static double tps[12][73]
            = { 324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6, 297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
01077
                       175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
01079
01080
                       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,
01081
01082
01083
           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,
01084
01085
01086
             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,
01087
01088
01089
01091
              284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01092
              287.5, 286.2, 285.8},
01093
            {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,
01094
01095
01096
              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,
01099
01100
            304.3, 304.9, 306, 306.6, 306.2, 306},
01101
           {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,
01102
01103
            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,
01105
01106
            148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
01107
            263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4
           205.7, 270.3, 270.6, 262.0, 260.1, 294.3, 300.4, 300.3, 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, 205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
01108
01109
01110
01111
01112
            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,
01113
01114
            273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01115
01116
            325.3, 325.8, 325.8},
           {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
            222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
01118
01119
            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,
01120
01121
01122
01123
            308.5, 312.2, 313.1, 313.3},
01124
01125
           {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,
01126
01127
01128
01129
            117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
01130
01131
            224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5
01132
            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,
01133
            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,
01134
01136
            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,
01137
01138
            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,
01139
           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,
01140
01141
            183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
01142
01143
            243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
01144
            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,
01145
01146
01147
            276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
01148
           {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
01149
01150
            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,
01151
01152
            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,
01153
            206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01155
01156
            279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
            305.1},
01157
           {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01158
           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,
01159
01160
            108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5,
01162
            102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
           109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4, 241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
01163
01164
            286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6}
01165
           {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
01166
            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,
01168
01169
            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,
01170
01171
01172
            280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
            281.7, 281.1, 281.2}
01173
01174
01175
01176
          double doy, p0, p1, pt;
01177
01178
          int imon, ilat;
          /* Get day of year... */
doy = fmod(t / 86400., 365.25);
01180
01181
01182
          while (doy < 0)
01183
             dov += 365.25;
01184
```

```
01185
         /* Get indices... */
         imon = locate(doys, 12, doy);
ilat = locate(lats, 73, lat);
01186
01187
01188
         /* Get tropopause pressure... */
01189
        p0 = LIN(lats[ilat], tps[imon][ilat],
lats[ilat + 1], tps[imon][ilat + 1], lat);
01190
01191
        p1 = LIN(lats[ilat], tps[imon + 1][ilat],
lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
01192
01193
01194
         pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01195
         /* Return tropopause pressure... */
01196
01197
         return pt;
01198 }
```

Here is the call graph for this function:



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

Write atmospheric data.

Definition at line 1202 of file libtrac.c.

```
01206
01207
01208
        FILE *in, *out;
01209
        char line[LEN];
01210
01211
01212
        double r;
01213
01214
        int ip, iq, year, mon, day, hour, min, sec;
01215
        /* Check if gnuplot output is requested... */
if (ctl->atm_gpfile[0] != '-') {
01216
01217
01218
01219
           /\star Write info... \star/
01220
          printf("Plot atmospheric data: %s.png\n", filename);
01221
          /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
01222
01223
             ERRMSG("Cannot create pipe to gnuplot!");
01224
01225
           01226
01227
01228
01229
           /* Set time string... */
           jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01230
01231
01232
                    year, mon, day, hour, min);
01233
01234
           /* Dump gnuplot file to pipe... */
           if (!(in = fopen(ctl->atm_gpfile, "r")))
ERRMSG("Cannot open file!");
01235
01236
           while (fgets(line, LEN, in))
fprintf(out, "%s", line);
01237
01238
01239
           fclose(in);
01240
01241
01242
        else {
01243
01244
          /* Write info... */
```

```
printf("Write atmospheric data: %s\n", filename);
01246
01247
           /* Create file... */
          if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");
01248
01249
01250
01251
01252
        /* Write header... */
01253
        fprintf(out,
                (Out,
"# $1 = time [s]\n"
"# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01254
01255
01256
        01257
01258
01259
        fprintf(out, "\n");
01260
01261
01262
        /* Write data... */
        01263
01264
01265
          for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01266
01267
01268
01269
01270
          fprintf(out, "\n");
01271
01272
01273
        /\star Close file... \star/
01274
        fclose(out);
01275 }
```

Here is the call graph for this function:



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

Write CSI data.

Definition at line 1279 of file libtrac.c.

```
01283
01284
       static FILE *in, *out;
01286
01287
       static char line[LEN];
01288
        static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01289
01290
         rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01291
01292
        static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01293
        /* Init... */
01294
01295
        if (!init) {
01296
         init = 1;
01297
01298
          /* Check quantity index for mass... */
01299
          if (ctl->qnt_m < 0)
           ERRMSG("Need quantity mass to analyze CSI!");
01300
01301
01302
          /* Open observation data file... */
         printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
01303
01304
          if (!(in = fopen(ctl->csi_obsfile, "r")))
```

```
01305
            ERRMSG("Cannot open file!");
01306
01307
          /* Create new file... */
          printf("Write CSI data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01308
01309
            ERRMSG("Cannot create file!");
01310
01311
01312
          /* Write header... */
01313
          fprintf(out,
                   "# $1 = time [s]\n"
"# $2 = number of hits (cx)\n"
01314
01315
                   "# $3 = number of misses (cy) \n"
01316
                   "# $4 = number of false alarms (cz)\n"
01317
01318
                   "# $5 = number of observations (cx + cy)\n"
01319
                   "# $6 = number of forecasts (cx + cz)n"
01320
                   "# \$7 = bias (forecasts/observations) [ \% ] \n"
                   "# $8 = probability of detection (POD) [%%]\n" "# $9 = false alarm rate (FAR) [%%]\n" "# $10 = critical success index (CSI) [%%]\n\n");
01321
01322
01323
01324
        }
01325
01326
        /* Set time interval... */
       t0 = t - 0.5 * ctl->dt_mod;

t1 = t + 0.5 * ctl->dt_mod;
01327
01328
01329
01330
        /* Initialize grid cells... */
01331
        for (ix = 0; ix < ctl->csi_nx; ix++)
01332
         for (iy = 0; iy < ctl->csi_ny; iy++)
01333
            for (iz = 0; iz < ctl->csi_nz; iz++)
01334
              modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01335
01336
        /* Read data...
01337
        while (fgets(line, LEN, in)) {
01338
           /* Read data... *,
01339
          if (sscanf(line, "%lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01340
01341
               5)
01342
             continue;
01343
01344
          /* Check time... */
01345
          <u>if</u> (rt < t0)
01346
            continue;
          if (rt > t1)
01347
01348
            break;
01349
01350
          /* Calculate indices... */
01351
          ix = (int) ((rlon - ctl->csi_lon0))
                        / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01352
          iy = (int) ((rlat - ctl->csi_lat0)
01353
01354
                        / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
          iz = (int) ((rz - ctl - > csi_z0))
01355
01356
                       / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01357
          /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
01358
01359
              iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01360
01361
             continue:
01362
01363
          /\star Get mean observation index... \star/
01364
          obsmean[ix][iy][iz] += robs;
01365
          obscount[ix][iy][iz]++;
01366
01367
01368
        /* Analyze model data... */
01369
        for (ip = 0; ip < atm->np; ip++) {
01370
01371
          /* Check time... */
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
01372
01373
            continue;
01374
01375
           /* Get indices... */
01376
          ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
                        / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01377
          01378
01379
          01380
01381
01382
          /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
    iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01383
01384
01385
01386
             continue;
01387
01388
           /∗ Get total mass in grid cell... ∗/
01389
          modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01390
01391
```

```
/* Analyze all grid cells... */
         for (ix = 0; ix < ctl->csi_nx; ix++)
01393
01394
           for (iy = 0; iy < ctl->csi_ny; iy++)
             for (iz = 0; iz < ctl->csi_nz; iz++) {
01395
01396
01397
                 /* Calculate mean observation index... */
                if (obscount[ix][iy][iz] > 0)
01398
01399
                   obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01400
01401
                /\star Calculate column density... \star/
                if (modmean[ix][iy][iz] > 0) {
01402
                  dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
01403
01404
01405
                   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);
01406
01407
01408
01409
                }
01410
01411
                /* Calculate CSI... */
01412
                if (obscount[ix][iy][iz] > 0) {
01413
                   if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
                       modmean[ix][iy][iz] >= ctl->csi_modmin)
01414
01415
                     cx++;
01416
                  else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
                              modmean[ix][iy][iz] < ctl->csi_modmin)
01417
01418
                     cy++;
01419
                   else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
                              modmean[ix][iy][iz] >= ctl->csi_modmin)
01420
01421
                     cz++;
01422
01423
01424
01425
         /* Write output... */
01426
         if (fmod(t, ctl->csi_dt_out) == 0) {
01427
           01428
01430
                     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);
01431
01432
01433
01434
01435
01436
            /* Set counters to zero... */
01437
           cx = cy = cz = 0;
01438
01439
         /* Close file... */
01440
01441
         if (t == ctl->t_stop)
01442
           fclose(out);
01443 }
```

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

Write gridded data.

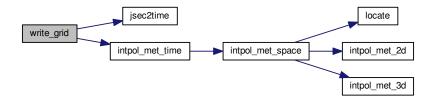
Definition at line 1447 of file libtrac.c.

```
01453
                  {
01454
01455
       FILE *in, *out;
01456
01457
       char line[LEN];
01458
       static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01459
01460
         area, rho_air, press, temp, cd, mmr, t0, t1, r;
01461
01462
       static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01463
01464
        /* Check dimensions... */
        if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
01465
         ERRMSG("Grid dimensions too large!");
01466
01467
        /\star Check quantity index for mass... \star/
01468
01469
       if (ctl->qnt_m < 0)
01470
         ERRMSG("Need quantity mass to write grid data!");
01471
01472
       /* Set time interval for output... */
01473
       t0 = t - 0.5 * ctl->dt_mod;
```

```
01474
        t1 = t + 0.5 * ctl->dt_mod;
01475
01476
         /* 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;
01477
01478
01479
01480
01481
          /* Initialize grid... */
01482
         for (ix = 0; ix < ctl->grid_nx; ix++)
            for (iy = 0; iy < ctl->grid_ny; iy++)
  for (iz = 0; iz < ctl->grid_nz; iz++)
01483
01484
01485
                 grid_m[ix][iy][iz] = 0;
01486
01487
          /* Average data... */
01488
         for (ip = 0; ip < atm->np; ip++)
01489
           if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {</pre>
01490
01491
               /* Get index... */
              ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
01492
              iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
01493
01494
              iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01495
              /* Check indices... */
if (ix < 0 || ix >= ctl->grid_nx ||
    iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01496
01497
01498
01499
                 continue:
01500
               /* Add mass... */
01501
              grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01502
01503
01504
         /* Check if gnuplot output is requested... */
if (ctl->grid_gpfile[0] != '-') {
01505
01506
01507
           /* Write info... */
printf("Plot grid data: %s.png\n", filename);
01508
01509
01510
01511
            /* Create gnuplot pipe... */
           if (!(out = popen("gnuplot", "w")))
01512
01513
             ERRMSG("Cannot create pipe to gnuplot!");
01514
           /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
01515
01516
01517
01518
            /* Set time string... */
           jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01519
01520
01521
                     year, mon, day, hour, min);
01522
01523
           /* Dump gnuplot file to pipe... */
           if (!(in = fopen(ctl->grid_gpfile, "r")))
    ERRMSG("Cannot open file!");
01524
01525
01526
            while (fgets(line, LEN, in))
01527
              fprintf(out, "%s", line);
01528
           fclose(in);
01529
         }
01530
01531
01532
01533
            /* Write info... */
           printf("Write grid data: %s\n", filename);
01534
01535
01536
            /* Create file... */
           if (!(out = fopen(filename, "w")))
01537
01538
              ERRMSG("Cannot create file!");
01539
01540
         /* Write header... */
01541
01542
         fprintf(out,
                   "# $1 = time [s] \n"
01544
                   "# $2 = altitude [km] \n"
                   "# $3 = longitude [deg] \n"
01545
                   "# $4 = latitude [deg]\n"
01546
                   "# $5 = surface area [km^2]\n"
"# $6 = layer width [km]\n"
01547
01548
01549
                   "# $7 = temperature [K] \n"
01550
                   "# $8 = \text{column density } [kg/m^2] \n"
                   "# $9 = mass mixing ratio [1]\n\n");
01551
01552
01553
         /* Write data... */
         for (ix = 0; ix < ctl->grid_nx; ix++) {
01554
          if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01556
              fprintf(out, "\n");
01557
            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++)
01558
01559
01560
```

```
if (!ctl->grid_sparse
01562
                      || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01563
                   /* Set coordinates... */
01564
                   z = ctl->grid_z0 + dz * (iz + 0.5);
lon = ctl->grid_lon0 + dlon * (ix + 0.5);
01565
01566
                   lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01567
01568
01569
                   /\star Get pressure and temperature... \star/
01570
                   press = P(z);
                   01571
01572
01573
01574
                    /* Calculate surface area... */
                   area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
 * cos(lat * M_PI / 180.);
01575
01576
01577
01578
                   /* Calculate column density... */
cd = grid_m[ix][iy][iz] / (1e6 * area);
01579
01580
                   /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01581
01582
01583
01584
01585
                    /* Write output... */
01586
                   fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g\n", t, z, lon, lat, area, dz, temp, cd, mmr);
01587
01588
01589
            }
01590
         }
01591
01592
          /* Close file... */
01593
         fclose(out);
01594 }
```

Here is the call graph for this function:



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

Write profile data.

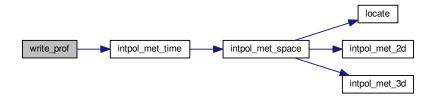
Definition at line 1598 of file libtrac.c.

```
01604
                    {
01605
01606
        static FILE *in, *out;
01607
01608
        static char line[LEN];
01609
        static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
01610
          rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, mmr, h2o, o3;
01611
01612
01613
01614
        static int init, obscount[GX][GY], ip, ix, iy, iz;
01615
01616
         /* Init... */
        if (!init) {
01617
01618
          init = 1;
01619
01620
          /* Check quantity index for mass... */
```

```
if (ctl->qnt_m < 0)
             ERRMSG("Need quantity mass!");
01622
01623
           /* Check dimensions... */
01624
           if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
    ERRMSG("Grid dimensions too large!");
01625
01626
01627
01628
           /* Open observation data file... */
01629
           printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01630
            if (!(in = fopen(ctl->prof_obsfile, "r")))
             ERRMSG("Cannot open file!");
01631
01632
           /* Create new file... */
01633
           printf("Write profile data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01634
01635
             ERRMSG("Cannot create file!");
01636
01637
01638
           /* Write header... */
01639
           fprintf(out,
                    "# $1
                            = time [s]\n"
01641
                    "# $2
                            = altitude [km]\n"
                            = longitude [deg]\n"
01642
                     "# $3
                    "# $4 = latitude [deg]\n"
01643
                     "# $5 = pressure [hPa] \n"
01644
01645
                    "# $6 = temperature [K]\n"
                    "# $7 = mass mixing ratio [1]\n"
01646
01647
                    "# $8 = H20 volume mixing ratio [1]\n"
01648
                    "# $9 = 03 volume mixing ratio [1]\n"
                     "# $10 = mean BT index [K] \n");
01649
01650
01651
           /* Set grid box size... */
01652
           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;
01653
01654
01655
01656
         /* Set time interval... */
01657
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01658
01660
01661
         /* Initialize... */
         for (ix = 0; ix < ctl->prof_nx; ix++)
01662
          for (iy = 0; iy < ctl->prof_ny; iy++) {
01663
             obsmean[ix][iy] = 0;
01664
              obscount[ix][iy] = 0;
01665
01666
              tmean[ix][iy] = 0;
01667
              for (iz = 0; iz < ctl->prof_nz; iz++)
01668
               mass[ix][iy][iz] = 0;
01669
01670
01671
         /* Read data... */
01672
         while (fgets(line, LEN, in)) {
01673
          /* Read data... */
if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01674
01675
01676
             continue;
01677
01678
           /* Check time... */
01679
           if (rt < t0)</pre>
01680
             continue;
           if (rt > t1)
01681
01682
             break;
01683
           /* Calculate indices... */
          ix = (int) ((rlon - ctl->prof_lon0) / dlon);
iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01685
01686
01687
           /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01688
01689
             continue;
01691
01692
           /\star Get mean observation index... \star/
           obsmean[ix][iy] += robs;
tmean[ix][iy] += rt;
01693
01694
           obscount[ix][iy]++;
01695
01696
01697
01698
         /* Analyze model data... */
01699
         for (ip = 0; ip < atm->np; ip++) {
01700
01701
           /* Check time... */
01702
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
01703
             continue;
01704
           /* Get indices... */
01705
           ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
01706
01707
```

```
iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01709
01710
            /* Check indices... */
           if (ix < 0 || ix >= ctl->prof_nx ||
    iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01711
01712
01713
              continue;
01714
01715
            /\star Get total mass in grid cell... \star/
01716
           mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01717
01718
01719
         /* Extract profiles... */
         for (ix = 0; ix < ctl->prof_nx; ix++)
  for (iy = 0; iy < ctl->prof_ny; iy++)
01720
01721
01722
              if (obscount[ix][iy] > 0) {
01723
                /* Write output... */
fprintf(out, "\n");
01724
01725
01726
01727
                 /* Loop over altitudes... */
01728
                 for (iz = 0; iz < ctl->prof_nz; iz++) {
01729
01730
                   /* Set coordinates... */
                   z = ctl->prof_z0 + dz * (iz + 0.5);
lon = ctl->prof_lon0 + dlon * (ix + 0.5);
01731
01732
01733
                   lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01734
01735
                   /\star Get meteorological data... \star/
01736
                   press = P(z);
                   01737
01738
01739
                   /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
01740
01741
                  area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
 * cos(lat * M_PI / 180.);
01742
01743
01744
                   mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
01745
01746
                   /* Write output... */
                   fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g \n", tmean[ix][iy] / obscount[ix][iy],
01747
01748
                             z, lon, lat, press, temp, mmr, h2o, o3,
obsmean[ix][iy] / obscount[ix][iy]);
01749
01750
01751
01752
01753
01754
         /* Close file... */
         if (t == ctl->t_stop)
01755
01756
            fclose(out):
01757 }
```

Here is the call graph for this function:



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

Write station data.

Definition at line 1761 of file libtrac.c.

```
01765
                    {
01766
01767
        static FILE *out;
01768
01769
        static double rmax2, t0, t1, x0[3], x1[3];
01770
01771
        static int init, ip, iq;
01772
01773
        /* Init... */
01774
        if (!init) {
01775
          init = 1;
01776
01777
          /* Write info... */
01778
          printf("Write station data: %s\n", filename);
01779
01780
           /\star Create new file... \star/
          if (!(out = fopen(filename, "w")))
01781
            ERRMSG("Cannot create file!");
01782
01783
01784
           /* Write header... */
01785
           fprintf(out,
                   "# $1 = time [s]\n"
"# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01786
01787
01788
          01789
01790
01791
01792
01793
           /\star Set geolocation and search radius... \star/
01794
01795
          geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
01796
          rmax2 = gsl_pow_2(ctl->stat_r);
01797
01798
        /* Set time interval for output... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01799
01800
01801
01803
         /* Loop over air parcels... */
01804
        for (ip = 0; ip < atm->np; ip++) {
01805
01806
           /* Check time... */
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
01807
01808
            continue;
01809
01810
           /\star Check station flag... \star/
01811
           if (ctl->qnt_stat >= 0)
01812
           if (atm->q[ctl->qnt_stat][ip])
01813
               continue:
01814
01815
           /* Get Cartesian coordinates... */
01816
          geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
01817
01818
           /\star Check horizontal distance... \star/
          if (DIST2(x0, x1) > rmax2)
01819
01820
            continue;
01822
           /* Set station flag... */
01823
          if (ctl->qnt_stat >= 0)
01824
            atm->q[ctl->qnt_stat][ip] = 1;
01825
          /* Write data... */
fprintf(out, "%.2f %g %g %g",
01826
01827
01828
                   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]);
01829
01830
01831
01832
01833
          fprintf(out, "\n");
01835
01836
        /* Close file... */
        if (t == ctl->t_stop)
01837
01838
          fclose(out);
01839 }
```

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.
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
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
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
```

5.12 libtrac.c 67

```
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(ctl, filename, met0);
00134
00135
        get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
    dt_met, filename);
00136
        read_met(ctl, filename, met1);
00137
00138
00139
       /\star Read new data for forward trajectories... \star/
00140
      if (t > met1->time && ct1->direction == 1) {
       memcpy(met0, met1, sizeof(met_t));
get_met_help(t, 1, metbase, ctl->dt_met, filename);
read_met(ctl, filename, met1);
00141
00142
00143
00144
00145
00146
       /* Read new data for backward trajectories... */
      if (t < met0->time && ctl->direction == -1) {
00147
       memcpy(met1, met0, sizeof(met_t));
get_met_help(t, -1, metbase, ctl->dt_met, filename);
00148
00149
00150
        read_met(ctl, filename, met0);
00151
00152 }
00153
```

```
00155
00156 void get_met_help(
00157
       double t,
00158
       int direct,
00159
       char *metbase,
       double dt_met,
00160
       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
       /* Decode time... */
00173
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]")
       aux10 = wp * (array[ix + 1][iy + 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,
```

5.12 libtrac.c 69

```
00242
        double *t,
00243
        double *u,
        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... */
if (met->lon[met->nx - 1] > 180 && lon < 0)
00253
00254
00255
          lon += 360;
00256
        /* \ {\tt Get indices...} \ */
00257
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
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
00309
                          w == NULL ? NULL : &w0.
                          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;
00325
        if (t != NULL)
00326
         *t = wt * (t0 - t1) + t1;
        if (u != NULL)
00327
00328
          *u = wt * (u0 - u1) + u1;
```

```
00329
       if (v != NULL)
       *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 *dav.
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;
       t0.tm_mday = 1;
00357
       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
       *year = t1->tm_year + 1900;
*mon = t1->tm_mon + 1;
00365
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
00391
             ihi = i;
00392
           else
00393
             ilo = i;
00394
       } else
        while (ihi > ilo + 1) {
00395
          i = (ihi + ilo) >> 1;
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(
00409
      const char *filename,
ctl_t * ctl,
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: %s\n", filename);
00424
00425
         /* Open file... */
         if (!(in = fopen(filename, "r")))
00426
          ERRMSG("Cannot open file!");
00427
00428
00429
         /* Read line... */
00430
         while (fgets(line, LEN, in)) {
00431
           /* Read data... */
TOK(line, tok, "%lg", atm->time[atm->np]);
TOK(NULL, tok, "%lg", atm->p[atm->np]);
TOK(NULL, tok, "%lg", atm->lon[atm->np]);
TOK(NULL, tok, "%lg", atm->lat[atm->np]);
for (iq = 0; iq < ctl->nq; iq+)
TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00432
00433
00434
00435
00436
00437
00438
00439
00440
            /* Convert altitude to pressure... */
           atm \rightarrow p[atm \rightarrow np] = P(atm \rightarrow p[atm \rightarrow np]);
00442
           /* Increment data point counter... */
if ((++atm->np) > NP)
    ERRMSG("Too many data points!");
00443
00444
00445
00446
00447
00448
         /* Close file... */
00449
         fclose(in);
00450
00451
         /* Check number of points... */
00452
         if (atm->np < 1)
00453
           ERRMSG("Can not read any data!");
00454 }
00455
00457
00458 void read_ctl(
00459
        const char *filename,
        int argc,
00461
         char *argv[],
00462
        ctl_t * ctl) {
00463
00464
        int ip, iq;
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;
00475
         ctl->qnt_ps = -1;
         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
             ctl->qnt_m = iq;
00496
              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");
00504
           } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
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");
00509
           } 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) {
              ctl->qnt_w = iq;
00517
           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
00529
              ctl->qnt_stat = iq;
00530
              sprintf(ctl->qnt_unit[iq], "-");
00531
             scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00532
00533
00534
00535
         /* Time steps of simulation... */
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...
         /* Netcetionstal data:.../
ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
00546
00547
00548
            (ctl->met_np > EP)
00549
          ERRMSG("Too many levels!");
         for (ip = 0; ip < ctl->met_np; ip++)
  ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
00550
00551
00552
00553
         /* Isosurface parameters... */
00554
         = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00555
00556
00557
00558
         /* Diffusion parameters... */
00559
         ctl->turb_dx_trop
00560
           = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00561
         ctl->turb_dx_strat
           = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00562
00563
         ctl->turb dz trop
00564
           = scan ctl(filename, argc, argv, "TURB DZ TROP", -1, "0.0", NULL);
00565
         ctl->turb_dz_strat
00566
            = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00567
         ctl->turb_meso =
00568
           scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00569
         /* Life time of particles... */
00570
         ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
00571
         ctl->tdec_strat =
00572
00573
           scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00574
        /* Output of atmospheric data... */
scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
00575
00576
      atm basename);
00577
        scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00578
         ctl->atm_dt_out =
00579
           scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00580
         /* Output of CSI data... */
00581
        scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
00582
      csi_basename);
00583 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",
00584
00585
00586
                   ctl->csi_obsfile);
00587
        ctl->csi_obsmin =
```

```
scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00589
         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);
00590
00591
00592
00593
00594
         ctl->csi lon0 =
00595
           scan_ctl(filename, argc, argv, "CSI_LONO", -1, "-180", NULL);
         ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
00596
         ctl->csi nx =
00597
        (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);
00598
00599
00600
00601
         ctl->csi_ny =
00602
           (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
00603
00604
        /* Output of grid data... */
        00605
00606
00607
         scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
      grid_gpfile);
00608
        ctl->grid_dt_out =
           scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
00609
00610
         ctl->grid sparse
           (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
00611
         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);
00612
00613
00614
         ctl->grid_nz =
00615
           (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
         ctl->grid_lon0 =
00616
00617
          scan_ctl(filename, argc, argv, "GRID_LONO", -1, "-180", NULL);
00618
        ctl->grid lon1
00619
           scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
00620
         ctl->grid_nx =
00621
           (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00622
         ctl->grid_lat0 =
           scan ctl(filename, argc, argv, "GRID LATO", -1, "-90", NULL);
00623
00624
         ctl->grid_lat1 =
00625
           scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
00626
         ctl->grid_ny =
00627
           (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
00628
00629
        /* Output of profile data... */
        00630
00631
00632
         scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
      prof_obsfile);
        ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL); ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
00633
00634
00635
         ctl->prof nz =
00636
           (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
         ctl->prof_lon0 =
00637
00638
           scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
00639
        ctl->prof_lon1
           scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
00640
00641
         ctl->prof nx =
           (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00642
00643
        ctl->prof lat0 =
00644
           scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
         ctl->prof_lat1
00645
          scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
00646
00647
        ctl->prof ny =
00648
           (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00649
00650
         /* Output of station data... */
        scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
00651
00652
                   ctl->stat basename);
        ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
00653
00654
00655
00656 }
00657
00659
00660 void read met (
00661
        ctl_t * ctl,
         char *filename,
00662
00663
         met_t * met) {
00664
00665
        char tstr[101:
00666
00667
        static float help[EX * EY];
00668
00669
        int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00670
00671
         size_t np, nx, ny;
00672
```

```
/* Write info... */
00674
        printf("Read meteorological data: %s\n", filename);
00675
        /* Get time from filename... */ sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00676
00677
00678
        year = atoi(tstr);
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00680
        mon = atoi(tstr);
00681
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00682
        day = atoi(tstr);
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00683
        hour = atoi(tstr);
00684
        time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00685
00686
        /* Open netCDF file... */
00687
00688
        NC(nc_open(filename, NC_NOWRITE, &ncid));
00689
        /* Get dimensions... */
NC(nc_inq_dimid(ncid, "lon", &dimid));
00690
00691
        NC(nc_inq_dimlen(ncid, dimid, &nx));
00692
00693
        if (nx > EX)
00694
          ERRMSG("Too many longitudes!");
00695
        NC(nc_inq_dimid(ncid, "lat", &dimid));
00696
00697
        NC(nc_inq_dimlen(ncid, dimid, &ny));
00698
        if (ny > EY)
00699
          ERRMSG("Too many latitudes!");
00700
        NC(nc_inq_dimid(ncid, "lev", &dimid));
NC(nc_inq_dimlen(ncid, dimid, &np));
00701
00702
00703
        if (np > EP)
00704
          ERRMSG("Too many levels!");
00705
00706
        /* Store dimensions... */
        met->np = (int) np;

met->nx = (int) nx;
00707
00708
00709
        met->ny = (int) ny;
00710
00711
         /* Get horizontal grid... */
00712
        NC(nc_inq_varid(ncid, "lon", &varid));
        NC(nc_get_var_double(ncid, varid, met->lon));
NC(nc_inq_varid(ncid, "lat", &varid));
00713
00714
00715
        NC(nc_get_var_double(ncid, varid, met->lat));
00716
        00717
00718
00719
00720
00721
00722
        read_met_help(ncid, "o3", "O3", met, met->o3, 0.602f);
00723
00724
00725
        /* Meteo data on pressure levels... */
00726
        if (ctl->met_np <= 0) {</pre>
00727
00728
          /* Read pressure levels from file... *
NC(nc_inq_varid(ncid, "lev", &varid));
00729
00730
          NC(nc_get_var_double(ncid, varid, met->p));
          for (ip = 0; ip < met->np; ip++)
  met->p[ip] /= 100.;
00731
00732
00733
00734
           /* Extrapolate data for lower boundary... */
00735
          read_met_extrapolate(met);
00736
00737
00738
        /\star Meteo data on model levels... \star/
00739
        else {
00740
00741
           /* Read pressure data from file... */
00742
          read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00743
00744
           /\star Interpolate from model levels to pressure levels... \star/
00745
          read_met_ml2pl(ctl, met, met->t);
00746
          read_met_ml2pl(ctl, met, met->u);
          read_met_ml2pl(ctl, met, met->v);
00747
00748
          read_met_ml2pl(ctl, met, met->w);
00749
           read_met_ml2pl(ctl, met, met->h2o);
00750
          read_met_ml2pl(ctl, met, met->o3);
00751
00752
           /* Set pressure levels... */
          met->np = ctl->met_np;
for (ip = 0; ip < met->np; ip++)
00753
00754
00755
            met->p[ip] = ctl->met_p[ip];
00756
00757
00758
        /* Check ordering of pressure levels... */
00759
        for (ip = 1; ip < met->np; ip++)
```

```
if (met->p[ip - 1] < met->p[ip])
00761
             ERRMSG("Pressure levels must be descending!");
00762
        /* Read surface pressure... */
if (nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
   NC(nc_get_var_float(ncid, varid, help));
00763
00764
00765
          for (iy = 0; iy < met->ny; iy++)
  for (ix = 0; ix < met->nx; ix++)
00766
00767
        met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
} else if (nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00768
00769
00770
          NC(nc_get_var_float(ncid, varid, help));
          for (iy = 0; iy < met->ny; iy++)
  for (ix = 0; ix < met->nx; ix++)
00771
00772
00773
               met \rightarrow ps[ix][iy] = exp(help[iy * met \rightarrow nx + ix]) / 100.;
00774
        } else
00775
          for (ix = 0; ix < met->nx; ix++)
             for (iy = 0; iy < met->ny; iy++)
  met->ps[ix][iy] = met->p[0];
00776
00777
00779
        /* Create periodic boundary conditions... */
00780
        read_met_periodic(met);
00781
00782
        /* Close file... */
00783
        NC(nc_close(ncid));
00784 }
00785
00787
00788 void read_met_extrapolate(
00789
        met_t * met) {
00790
00791
        int ip, ip0, ix, iv;
00792
        /* Loop over columns... */
00793
00794
        for (ix = 0; ix < met->nx; ix++)
00795
          for (iy = 0; iy < met->ny; iy++) {
00796
             /* Find lowest valid data point... */
00798
             for (ip0 = met -> np - 1; ip0 >= 0; ip0 --)
00799
               if (!gsl_finite(met->t[ix][iy][ip0])
00800
                    || !gsl_finite(met->u[ix][iy][ip0])
00801
                    || !gsl_finite(met->v[ix][iy][ip0])
                    || !gsl_finite(met->w[ix][iy][ip0]))
00802
00803
                 break;
00804
00805
             /* Extrapolate... */
00806
             for (ip = ip0; ip >= 0; ip--) {
              met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
00807
               met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
00808
00809
               met \rightarrow v[ix][iy][ip] = met \rightarrow v[ix][iy][ip + 1];
               met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
00810
               met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00811
00812
00813
00814
00815 }
00818
00819 void read_met_help(
00820
        int ncid,
00821
        char *varname,
00822
        char *varname2,
00823
        met_t * met,
00824
        float dest[EX][EY][EP],
00825
        float scl) {
00826
00827
        static float help[EX * EY * EP];
00828
        int ip, ix, iy, n = 0, varid;
00830
00831
        /\star Check if variable exists... \star/
00832
        if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
          if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00833
00834
             return;
00835
00836
         /* Read data... */
00837
        NC(nc_get_var_float(ncid, varid, help));
00838
00839
         /* Copy and check data... */
00840
        for (ip = 0; ip < met->np; ip++)
          for (iy = 0; iy < met->ny; iy++)
00841
             for (ix = 0; ix < met->nx; ix++) {
  dest[ix][iy][ip] = scl * help[n++];
00842
00843
               if (fabs(dest[ix][iy][ip] / scl) > 1e14)
00844
00845
                 dest[ix][iy][ip] = GSL_NAN;
00846
```

```
00847 }
00848
       00849 /
00850
00851 void read met ml2pl(
       ctl_t * ctl,
met_t * met,
00852
00853
00854
       float var[EX][EY][EP]) {
00855
00856
       double aux[EP], p[EP], pt;
00857
00858
       int ip, ip2, ix, iy;
00859
00860
        /* Loop over columns... */
00861
       for (ix = 0; ix < met->nx; ix++)
00862
         for (iy = 0; iy < met->ny; iy++) {
00863
00864
            /* Copy pressure profile... */
            for (ip = 0; ip < met->np; ip++)
00865
00866
             p[ip] = met->pl[ix][iy][ip];
00867
00868
            /* Interpolate... */
           for (ip = 0; ip < ctl->met_np; ip++) {
00869
             pt = ctl->met_p[ip];
if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
00870
00871
00872
               pt = p[0];
00873
             else if ((pt > p[met->np - 1] && p[1] > p[0])
00874
                      || (pt < p[met->np - 1] && p[1] < p[0])
             00875
00876
00877
00878
00879
00880
           /* Copy data... */
for (ip = 0; ip < ctl->met_np; ip++)
00881
00882
             var[ix][iy][ip] = (float) aux[ip];
00883
00884
00885 }
00886
00888
00889 void read_met_periodic(
00890
       met_t * met) {
00891
00892
       int ip, iy;
00893
00894
       /* Check longitudes... */
       if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
00895
00896
                  + \text{ met} - > \text{lon}[1] - \text{ met} - > \text{lon}[0] - 360) < 0.01))
00897
         return;
00898
00899
       /* Increase longitude counter... */
00900
       if ((++met->nx) > EX)
         ERRMSG("Cannot create periodic boundary conditions!");
00901
00902
00903
       /* Set longitude... */
00904
       met \rightarrow lon[met \rightarrow nx - 1] = met \rightarrow lon[met \rightarrow nx - 2] + met \rightarrow lon[1] - met \rightarrow
     lon[0];
00905
       00906
00907
00908
00909
00910
00911
           met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
00912
           met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
00913
00914
           met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
00915
00916
00917 }
00918
       ******************************
00919 /
00920
00921 double scan_ctl(
00922
       const char *filename,
00923
       int argc,
       char *argv[],
const char *varname,
00924
00925
       int arridx,
const char *defvalue,
00926
00927
00928
       char *value) {
00929
00930
       FILE *in = NULL;
00931
00932
       char dummy [LEN], fullname1 [LEN], fullname2 [LEN], line [LEN],
```

```
msg[LEN], rvarname[LEN], rval[LEN];
00934
00935
        int contain = 0, i;
00936
        /* Open file... */
00937
        if (filename[strlen(filename) - 1] != '-')
00938
         if (!(in = fopen(filename, "r")))
00940
             ERRMSG("Cannot open file!");
00941
00942
        /\star Set full variable name... \star/
        if (arridx >= 0) {
00943
         sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
00944
00945
00946
         sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
00947
00948
00949
00950
00951
        /* Read data... */
00952
        if (in != NULL)
         while (fgets(line, LEN, in))
if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
if (strcasecmp(rvarname, fullname1) == 0 ||
00953
00954
00955
                   strcasecmp(rvarname, fullname2) == 0) {
00956
00957
                 contain = 1;
00958
                break;
00959
00960
        for (i = 1; i < argc - 1; i++)</pre>
         if (strcasecmp(argv[i], fullname1) == 0 ||
00961
            strcasecmp(argv[i], fullname2) == 0) {
sprintf(rval, "%s", argv[i + 1]);
00962
00963
00964
             contain = 1;
00965
            break;
00966
00967
        /* Close file... */
00968
        if (in != NULL)
00969
00970
          fclose(in);
00971
00972
        /* Check for missing variables... */
00973
        if (!contain) {
         if (strlen(defvalue) > 0)
00974
            sprintf(rval, "%s", defvalue);
00975
00976
          else {
00977
            sprintf(msg, "Missing variable %s!\n", fullname1);
00978
            ERRMSG(msg);
00979
00980
00981
00982
        /* Write info... */
        printf("%s = %s\n", fullname1, rval);
00983
00984
00985
        /* Return values... */
        if (value != NULL)
    sprintf(value, "%s", rval);
00986
00987
00988
        return atof(rval);
00990
00992
00993 void time2jsec(
00994
        int year,
00995
        int mon,
00996
        int day,
00997
        int hour,
00998
        int min,
00999
        int sec,
        double remain,
01000
        double *jsec) {
01001
01003
        struct tm t0, t1;
01004
01005
        t0.tm_year = 100;
        t0.tm_mon = 0;
01006
        t0.tm_mday = 1;
t0.tm_hour = 0;
01007
01008
01009
        t0.tm_min = 0;
01010
        t0.tm\_sec = 0;
01011
01012
        t1.tm year = year - 1900;
        t1.tm_mon = mon - 1;
01013
01014
        t1.tm_mday = day;
01015
        t1.tm_hour = hour;
        t1.tm_min = min;
01016
01017
        t1.tm_sec = sec;
01018
01019
        *isec = (double) timeqm(&t1) - (double) timeqm(&t0) + remain;
```

```
01021
01023
01024 void timer(
01025
         const char *name.
         int id,
01026
01027
         int mode) {
01028
01029
         static double starttime[NTIMER], runtime[NTIMER];
01030
         /* Check id... */
if (id < 0 || id >= NTIMER)
01031
01032
            ERRMSG("Too many timers!");
01033
01034
01035
          /* Start timer...
01036
         if (mode == 1) {
           if (starttime[id] <= 0)</pre>
01037
             starttime[id] = omp_get_wtime();
01039
01040
              ERRMSG("Timer already started!");
01041
01042
         /* Stop timer... */
else if (mode == 2) {
01043
01044
          if (starttime[id] > 0) {
01046
              runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
01047
              starttime[id] = -1;
01048
01049
              ERRMSG("Timer not started!");
01050
01051
01052
          /* Print timer... */
01053
         else if (mode == 3)
01054
           printf("%s = %g s\n", name, runtime[id]);
01055 }
01056
01058
01059 double tropopause(
01060
         double t
01061
         double lat) {
01062
01063
         static double doys[12]
         = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01064
01065
01066
         static double lats[73]
           01067
01068
01069
01071
           15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
01072
            45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
01073
           75, 77.5, 80, 82.5, 85, 87.5, 90
01074
01075
         static double tps[12][73]
           = { (324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
01077
01078
                  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
01079
01080
01081
01082
                  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,
01083
01084
                  275.3, 275.6, 275.4, 274.1, 273.5},
         {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7, 300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5, 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,
01085
01086
01087
01088
           98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
01090
           220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
01091
           284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
         287.5, 286.2, 285.8}, {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
01092
01093
          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,
01094
           100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01096
          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,
01097
01098
01099
           304.3, 304.9, 306, 306.6, 306.2, 306},
01100
          {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,
01102
01103
01104
           102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
          99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1, 148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
01105
01106
```

```
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,
01108
01109
                  260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
01110
                 205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7, 101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9, 102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9, 165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
01111
01112
01113
01114
01115
                  273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01116
                  325.3, 325.8, 325.8},
                (220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3, 222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2, 228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
01117
01118
01119
                 105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
01120
01121
                 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,
01122
01123
                308.5, 312.2, 313.1, 313.3, {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
01124
                  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,
01127
01128
                 110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
                 111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4, 117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9, 224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
01129
01130
01131
                  275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
01132
                {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01133
01134
                 185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
01135
                 233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
                 110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9, 112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3, 120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
01136
01137
01138
                  230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
01139
01140
                 278.2, 282.6, 287.4, 290.9, 292.5, 293},
01141
                {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
01142
                 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, 114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
01143
01145
                  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,
01146
               203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5, 276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1}, {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5, 215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
01147
01148
01149
01150
                 237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
01152
                 111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
                106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2, 112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7, 206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3, 279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
01153
01154
01155
01156
                  305.1},
                {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01158
01159
                  253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
01160
                 223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
                 108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
01161
                 102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2, 109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
01162
                  241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
01164
                 286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6}
01165
               280.1, 287.4, 288.3, 289.5, 290.9, 294.2, 290.9, 297.5, 297.5), (301.2, 300.3, 296.6, 295.4, 295., 294.3, 291.2, 287.4, 284.9, 284.7, 284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192, 175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1, 100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2, 100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160, 186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2, 200.2
01166
01167
01168
01169
01170
01171
01172
                 280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
01173
                 281.7, 281.1, 281.2}
01174
01175
               double doy, p0, p1, pt;
01177
01178
               int imon, ilat;
01179
               /* Get day of year... */
doy = fmod(t / 86400., 365.25);
while (doy < 0)
01180
01181
01182
01183
                   doy += 365.25;
01184
01185
               /* Get indices... */
               imon = locate(doys, 12, doy);
ilat = locate(lats, 73, lat);
01186
01187
01189
                /* Get tropopause pressure... */
01190
               p0 = LIN(lats[ilat], tps[imon][ilat],
01191
                                 lats[ilat + 1], tps[imon][ilat + 1], lat);
               01192
01193
```

```
pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01195
01196
        /* Return tropopause pressure... */
01197
        return pt;
01198 }
01199
01201
01202 void write_atm(
01203
        const char *filename,
        ctl_t * ctl,
atm_t * atm,
01204
01205
01206
        double t) {
01207
01208
       FILE *in, *out;
01209
        char line[LEN];
01210
01211
01212
        double r;
01213
01214
        int ip, iq, year, mon, day, hour, min, sec;
01215
        /* Check if gnuplot output is requested... */
if (ctl->atm_gpfile[0] != '-') {
01216
01217
01218
01219
           /* Write info... */
01220
          printf("Plot atmospheric data: %s.png\n", filename);
01221
          /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
01222
01223
            ERRMSG("Cannot create pipe to gnuplot!");
01224
01225
          /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
01226
01227
01228
01229
           /* Set time string... */
          jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01230
01231
01232
                   year, mon, day, hour, min);
01233
01234
           /\star Dump gnuplot file to pipe... \star/
          if (!(in = fopen(ctl->atm_gpfile, "r")))
    ERRMSG("Cannot open file!");
01235
01236
          while (fgets(line, LEN, in))
fprintf(out, "%s", line);
01237
01238
01239
          fclose(in);
01240
01241
01242
        else {
01243
01244
           /* Write info... */
01245
          printf("Write atmospheric data: %s\n", filename);
01246
01247
           /* Create file... */
          if (!(out = fopen(filename, "w")))
01248
            ERRMSG("Cannot create file!");
01249
01250
01251
01252
         /* Write header... */
        fprintf(out,
    "# $1 = time [s]\n"
01253
01254
                 "# $2 = altitude [km] \n"
01255
01256
                 "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
        for (iq = 0; iq < ctl->nq; iq++)
fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
01257
01258
        ctl->qnt_unit[iq]);
fprintf(out, "\n");
01259
01260
01261
01262
         /* Write data... */
        for (ip = 0; ip < atm->np; ip++) {
  fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
01263
01264
                   atm->lon[ip], atm->lat[ip]);
01265
          for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01266
01267
01268
             fprintf(out, ctl->qnt format[iq], atm->q[iq][ip]);
01269
01270
          fprintf(out, "\n");
01271
01272
01273
        /* Close file... */
01274
        fclose(out);
01276
01278
01279 void write csi(
01280
       const char *filename.
```

```
01281
        ctl_t * ctl,
01282
         atm_t * atm,
01283
        double t) {
01284
01285
         static FILE *in, *out;
01286
01287
        static char line[LEN];
01288
01289
        static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01290
           rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01291
        static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01292
01293
01294
01295
         if (!init) {
01296
          init = 1;
01297
           /* Check quantity index for mass... */
if (ctl->qnt_m < 0)</pre>
01298
01299
01300
             ERRMSG("Need quantity mass to analyze CSI!");
01301
01302
           /\star Open observation data file... \star/
           printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
if (!(in = fopen(ctl->csi_obsfile, "r")))
01303
01304
01305
             ERRMSG("Cannot open file!");
01306
           /\star Create new file... \star/
01307
           printf("Write CSI data: %s\n", filename);
if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");
01308
01309
01310
01311
01312
            /* Write header... */
01313
           fprintf(out,
01314
                    "# $1 = time [s] \n"
                    "# $2 = number of hits (cx) \n"
01315
                     "# $3 = number of misses (cy) \n"
01316
                    "# $4 = number of false alarms (cz)\n"
01317
                    "# $5 = number of observations (cx + cy) n"
01318
01319
                    "# $6 = number of forecasts (cx + cz)\n"
01320
                    "# $7 = bias (forecasts/observations) [%%]\n"
                    "# $8 = probability of detection (POD) [%%]\n" # $9 = false alarm rate (FAR) [%%]\n"
01321
01322
                    "# $10 = critical success index (CSI) [%%]\n\n");
01323
01324
01325
01326
         /* Set time interval... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01327
01328
01329
01330
         /* Initialize grid cells... */
01331
         for (ix = 0; ix < ctl->csi_nx; ix++)
01332
           for (iy = 0; iy < ctl->csi_ny; iy++)
01333
              for (iz = 0; iz < ctl->csi_nz; iz++)
01334
               modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01335
01336
         /* Read data... */
01337
         while (fgets(line, LEN, in)) {
01338
           /* Read data... */
if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01339
01340
01341
                5)
01342
             continue;
01343
01344
           /* Check time... */
01345
           if (rt < t0)
           continue;
if (rt > t1)
01346
01347
01348
             break:
01349
01350
           /* Calculate indices... */
01351
           ix = (int) ((rlon - ctl->csi\_lon0))
01352
                         / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
           iy = (int) ((rlat - ctl->csi_lat0))
01353
                         / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01354
           iz = (int) ((rz - ctl -> csi_z0))
01355
01356
                         / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01357
01358
            /* Check indices... */
           if (ix < 0 || ix >= ctl->csi_nx ||
    iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01359
01360
01361
             continue;
01362
01363
            /* Get mean observation index... */
01364
           obsmean[ix][iy][iz] += robs;
01365
           obscount[ix][iy][iz]++;
01366
01367
```

```
/* Analyze model data... */
01369
       for (ip = 0; ip < atm->np; ip++) {
01370
01371
          /* Check time... */
01372
         if (atm->time[ip] < t0 || atm->time[ip] > t1)
01373
           continue;
01374
01375
          /* Get indices... */
01376
         ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01377
                      / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
         iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
01378
01379
                      / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
         01380
01381
01382
          /* Check indices... */
01383
         if (ix < 0 || ix >= ctl->csi_nx ||
    iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01384
01385
01386
            continue;
01387
01388
          /* Get total mass in grid cell... */
01389
         modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01390
01391
01392
        /* Analyze all grid cells... */
       for (ix = 0; ix < ctl->csi_nx; ix++)
01393
01394
         for (iy = 0; iy < ctl->csi_ny; iy++)
01395
            for (iz = 0; iz < ctl->csi_nz; iz++) {
01396
01397
              /* Calculate mean observation index... */
01398
              if (obscount[ix][iv][iz] > 0)
01399
               obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01400
01401
              /\star Calculate column density... \star/
             if (modmean[ix][iy][iz] > 0) {
  dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
  dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
01402
01403
01404
               lat = ctl -> csi_lat0 + dlat * (iy + 0.5);
               area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180. * cos(lat * M_PI / 180.);
01406
01407
01408
               modmean[ix][iy][iz] /= (1e6 * area);
             }
01409
01410
01411
              /* Calculate CSI... */
              if (obscount[ix][iy][iz] > 0) {
01412
01413
               if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01414
                   modmean[ix][iy][iz] >= ctl->csi_modmin)
                 cx++;
01415
01416
               else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
                        modmean[ix][iy][iz] < ctl->csi_modmin)
01417
01418
                 су++;
01419
                else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
01420
                         modmean[ix][iy][iz] >= ctl->csi_modmin)
01421
                 cz++:
01422
           }
01423
01425
       /* Write output... */
01426
       if (fmod(t, ctl->csi_dt_out) == 0) {
01427
         /* Write... */
fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
01428
01429
                 01430
01432
01433
01434
01435
01436
          /* Set counters to zero... */
         cx = cy = cz = 0;
01437
01438
01439
01440
       /* Close file... */
       if (t == ctl->t_stop)
01441
01442
         fclose(out);
01443 }
01444
01446
01447 void write grid(
01448 const char *filename,
01449
       ctl_t * ctl,
01450
01451
       met_t * met1,
       atm t * atm,
01452
01453
       double t) {
01454
```

```
01455
        FILE *in, *out;
01456
01457
         char line[LEN];
01458
         static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01459
01460
           area, rho_air, press, temp, cd, mmr, t0, t1, r;
01461
01462
         static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01463
         /* Check dimensions... */
if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
    ERRMSG("Grid dimensions too large!");
01464
01465
01466
01467
01468
         /* Check quantity index for mass... */
01469
         if (ctl->qnt_m < 0)</pre>
01470
          ERRMSG("Need quantity mass to write grid data!");
01471
01472
         /* Set time interval for output... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01473
01474
01475
         /* Set grid box size... */
01476
        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;
01477
01478
01479
01480
01481
         /* Initialize grid... */
01482
         for (ix = 0; ix < ctl->grid_nx; ix++)
           for (iy = 0; iy < ctl->grid_ny; iy++)
  for (iz = 0; iz < ctl->grid_nz; iz++)
01483
01484
01485
                qrid_m[ix][iy][iz] = 0;
01486
01487
01488
         for (ip = 0; ip < atm->np; ip++)
01489
           if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {</pre>
01490
01491
              /* Get index... */
             ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
01492
01493
              iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
01494
             iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01495
              /* Check indices... */
01496
             /* check indices... */
if (ix < 0 || ix >= ctl->grid_nx ||
    iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01497
01498
01499
                continue;
01500
01501
              /* Add mass... */
             grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01502
01503
01504
01505
         /* Check if gnuplot output is requested... */
01506
         if (ctl->grid_gpfile[0] != '-') {
01507
01508
           /\star Write info... \star/
           printf("Plot grid data: %s.png\n", filename);
01509
01510
01511
           /* Create gnuplot pipe... */
01512
           if (!(out = popen("gnuplot", "w")))
01513
             ERRMSG("Cannot create pipe to gnuplot!");
01514
01515
           /\star Set plot filename... \star/
           fprintf(out, "set out \"%s.png\"\n", filename);
01516
01517
01518
           /* Set time string... */
01519
            jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
           fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01520
01521
                    year, mon, day, hour, min);
01522
01523
           /* Dump gnuplot file to pipe... */
           if (!(in = fopen(ctl->grid_gpfile, "r")))
             ERRMSG("Cannot open file!");
01525
01526
           while (fgets(line, LEN, in))
             fprintf(out, "%s", line);
01527
01528
           fclose(in);
01529
01530
01531
01532
01533
           /* Write info... */
          printf("Write grid data: %s\n", filename);
01534
01535
01536
           /* Create file... */
01537
           if (!(out = fopen(filename, "w")))
01538
             ERRMSG("Cannot create file!");
01539
01540
01541
        /* Write header... */
```

```
fprintf(out,
01543
                  "# $1 = time [s] \n"
                  "# $2 = altitude [km]\n"
01544
                  "# $3 = longitude [deg] \n"
01545
                  "# $4 = latitude [deg] \n"
01546
                  "# $5 = surface area [km^2]\n"
01547
                  "# $6 = layer width [km] \n"
01548
01549
                  "# $7 = temperature [K]\n"
01550
                  "# $8 = column density [kg/m^2]\n"
                  "# $9 = mass mixing ratio [1]\n\n");
01551
01552
        /* Write data... */
for (ix = 0; ix < ctl->grid_nx; ix++) {
   if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01553
01554
01555
           fprintf(out, "\n");
for (iy = 0; iy < ctl->grid_ny; iy++) {
   if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
01556
01557
01558
              fprintf(out, "\n");

for (iz = 0; iz < ctl->grid_nz; iz++)
01559
01560
               if (!ctl->grid_sparse
01562
                    || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01563
01564
                  /* Set coordinates... */
                  z = ctl->grid_z0 + dz * (iz + 0.5);
lon = ctl->grid_lon0 + dlon * (ix + 0.5);
01565
01566
                  lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01567
01568
                  /* Get pressure and temperature... */
press = P(z);
01569
01570
                  01571
01572
01574
                  /* Calculate surface area... */
                  area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
 * cos(lat * M_PI / 180.);
01575
01576
01577
01578
                  /* Calculate column density... */
01579
                  cd = grid_m[ix][iy][iz] / (1e6 * area);
01580
                  /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01581
01582
01583
01584
01585
                  /* Write output... */
                  fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n", t, z, lon, lat, area, dz, temp, cd, mmr);
01586
01587
01588
        }
01589
01590
01591
01592
         /* Close file... */
01593
        fclose(out);
01594 }
01595
01597
01598 void write_prof(
        const char *filename,
01599
01600
         ctl_t * ctl,
01601
         met_t * met0,
        met_t * met1,
01602
        atm_t * atm,
01603
01604
        double t) {
01605
01606
        static FILE *in, *out;
01607
01608
        static char line[LEN];
01609
        static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
  rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z,
01610
01611
01612
          press, temp, rho_air, mmr, h2o, o3;
01613
01614
        static int init, obscount[GX][GY], ip, ix, iy, iz;
01615
01616
         /* Init... */
01617
         if (!init) {
01618
           init = 1;
01619
           /* Check quantity index for mass... */
if (ctl->qnt_m < 0)</pre>
01620
01621
             ERRMSG("Need quantity mass!");
01622
01623
01624
           /* Check dimensions...
           if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
01625
01626
             ERRMSG("Grid dimensions too large!");
01627
01628
           /* Open observation data file... */
```

```
printf("Read profile observation data: %s\n", ctl->prof_obsfile);
             f (!(in = fopen(ctl->prof_obsfile, "r")))
ERRMSG("Cannot open file!");
01630
01631
01632
01633
           /* Create new file... */
printf("Write profile data: %s\n", filename);
01634
           if (!(out = fopen(filename, "w")))
01635
01636
             ERRMSG("Cannot create file!");
01637
01638
           /* Write header... */
01639
           fprintf(out,
                     "# $1
                            = time [s]\n"
01640
                    "# $2
                            = altitude [km]\n"
01641
01642
                    "# $3
                            = longitude [deg]\n"
01643
                    "# $4
                            = latitude [deg]\n"
01644
                     "# $5 = pressure [hPa] \n"
                     "# $6 = temperature [K]\n"
01645
                     "# $7 = mass mixing ratio [1]\n"
01646
                     "# $8 = H2O volume mixing ratio [1]\n"
01647
                     "# $9 = O3 volume mixing ratio [1]\n"
01648
01649
                     "# $10 = mean BT index [K]\n");
01650
           /\star Set grid box size... \star/
01651
           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;
01652
01653
01654
01655
01656
        /* Set time interval... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01657
01658
01659
01660
01661
         /* Initialize... */
01662
         for (ix = 0; ix < ctl->prof_nx; ix++)
01663
           for (iy = 0; iy < ctl->prof_ny; iy++) {
             obsmean[ix][iy] = 0;
01664
             obscount[ix][iy] = 0;
tmean[ix][iy] = 0;
01665
01666
01667
              for (iz = 0; iz < ctl->prof_nz; iz++)
01668
               mass[ix][iy][iz] = 0;
01669
01670
01671
        /* Read data... */
        while (fgets(line, LEN, in)) {
01672
01673
01674
           /* Read data... */
01675
           if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
             continue;
01676
01677
01678
           /* Check time... */
01679
           if (rt < t0)</pre>
01680
             continue;
           if (rt > t1)
01681
01682
             break;
01683
01684
           /* Calculate indices... */
           ix = (int) ((rlon - ctl->prof_lon0) / dlon);
01685
01686
           iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01687
           /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01688
01689
01690
             continue;
01691
01692
            /\star Get mean observation index... \star/
01693
           obsmean[ix][iy] += robs;
01694
           tmean[ix][iy] += rt;
01695
           obscount[ix][iy]++;
01696
01697
         /* Analyze model data... */
01699
         for (ip = 0; ip < atm->np; ip++) {
01700
            /* Check time... */
01701
01702
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
01703
             continue;
01704
01705
           ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
01706
01707
           iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01708
01709
           /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx ||
01711
01712
                iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01713
             continue;
01714
01715
           /* Get total mass in grid cell... */
```

```
01716
          mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01717
01718
01719
        /* Extract profiles... */
        for (ix = 0; ix < ctl->prof_nx; ix++)
  for (iy = 0; iy < ctl->prof_ny; iy++)
01720
01721
01722
            if (obscount[ix][iy] > 0) {
01723
               /* Write output...
01724
               fprintf(out, "\n");
01725
01726
               /* Loop over altitudes... */
01727
               for (iz = 0; iz < ctl->prof_nz; iz++) {
01728
01729
01730
                 /* Set coordinates... */
                z = ctl->prof_z0 + dz * (iz + 0.5);
lon = ctl->prof_lon0 + dlon * (ix + 0.5);
lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01731
01732
01733
01735
                 /* Get meteorological data... */
                 press = P(z);
01736
01737
                 intpol_met_time(met0, met1, t, press, lon, lat,
                                  NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
01738
01739
01740
                 /\star Calculate mass mixing ratio... \star/
01741
                rho_air = 100. * press / (287.058 * temp);
                area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
 * cos(lat * M_PI / 180.);
01742
01743
01744
                mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
01745
                 /* Write output... */
fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g,n",
01746
01747
01748
                          tmean[ix][iy] / obscount[ix][iy],
01749
                          z, lon, lat, press, temp, mmr, h2o, o3,
01750
                          obsmean[ix][iy] / obscount[ix][iy]);
01751
             }
01752
01753
        /* Close file... */
01754
01755
        if (t == ctl->t_stop)
01756
          fclose(out);
01757 }
01758
01760
01761 void write_station(
01762 const char *filename,
01763
        ctl_t * ctl,
        atm_t * atm,
01764
01765
        double t) {
01766
01767
        static FILE *out;
01768
01769
        static double rmax2, t0, t1, x0[3], x1[3];
01770
01771
        static int init, ip, iq;
01772
01773
        /* Init... */
01774
        if (!init) {
01775
          init = 1;
01776
          /* Write info... */
printf("Write station data: %s\n", filename);
01777
01778
01779
01780
           /* Create new file... */
          if (!(out = fopen(filename, "w")))
01781
            ERRMSG("Cannot create file!");
01782
01783
01784
           /* Write header... */
01785
          fprintf(out,
01786
                   "# $1 = time [s] \n"
                   "# $2 = altitude [km] \n"
01787
                   "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01788
           for (iq = 0; iq < ctl->nq; iq++)
  fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
01789
01790
01791
                     ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01792
          fprintf(out, "\n");
01793
01794
           /\star Set geolocation and search radius... \star/
01795
          geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
01796
          rmax2 = gsl_pow_2(ctl->stat_r);
01797
01798
01799
        /* Set time interval for output... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01800
01801
01802
```

```
/* Loop over air parcels... */
       for (ip = 0; ip < atm->np; ip++) {
01805
01806
          /\star Check time... \star/
01807
         if (atm->time[ip] < t0 || atm->time[ip] > t1)
01808
           continue;
01809
01810
         /* Check station flag... */
01811
         if (ctl->qnt_stat >= 0)
01812
          if (atm->q[ctl->qnt_stat][ip])
01813
             continue;
01814
01815
         /* Get Cartesian coordinates... */
01816
         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
01817
01818
          /\star Check horizontal distance... \star/
         if (DIST2(x0, x1) > rmax2)
01819
01820
           continue;
01821
01822
          /* Set station flag... */
01823
         if (ctl->qnt_stat >= 0)
01824
           atm->q[ctl->qnt_stat][ip] = 1;
01825
         01826
01827
01828
          for (iq = 0; iq < ctl->nq, iq++) {
  fprintf(out, " ");
  fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01829
01830
01831
01832
01833
         fprintf(out, "\n");
01834
01835
01836
       /* Close file... */
01837
       if (t == ctl->t_stop)
01838
         fclose(out);
01839 }
```

#### 5.13 libtrac.h File Reference

MPTRAC library declarations.

### **Data Structures**

• struct ctl t

Control parameters.

• struct atm\_t

Atmospheric data.

struct met t

Meteorological data.

## **Functions**

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

Convert Cartesian coordinates to geolocation.

double deg2dx (double dlon, double lat)

Convert degrees to horizontal distance.

• double deg2dy (double dlat)

Convert degrees to horizontal distance.

• double dp2dz (double dp, double p)

Convert pressure to vertical distance.

double dx2deg (double dx, double lat)

Convert horizontal distance to degrees.

double dy2deg (double dy)

Convert horizontal distance to degrees.

• double dz2dp (double dz, double p)

Convert vertical distance to pressure.

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

Convert geolocation to Cartesian coordinates.

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

Get meteorological data for given timestep.

• void get met help (double t, int direct, char \*metbase, double dt met, char \*filename)

Get meteorological data for timestep.

• void intpol\_met\_2d (double array[EX][EY], int ix, int iy, double wx, double wy, double \*var)

Linear interpolation of 2-D meteorological data.

- void intpol\_met\_3d (float array[EX][EY][EP], int ip, int ix, int iy, double wp, double wx, double wy, double \*var)

  Linear interpolation of 3-D meteorological data.
- void intpol\_met\_space (met\_t \*met, double p, double lon, double lat, double \*ps, double \*t, double \*u, double \*v, double \*w, double \*h2o, double \*o3)

Spatial interpolation of meteorological data.

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

Temporal interpolation of meteorological data.

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

Convert seconds to date.

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

Find array index.

void read atm (const char \*filename, ctl t \*ctl, atm t \*atm)

Read atmospheric data.

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

Read control parameters.

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

Read meteorological data file.

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

Extrapolate meteorological data at lower boundary.

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

  Read and convert variable from meteorological data file.
- void read\_met\_ml2pl (ctl\_t \*ctl, met\_t \*met, float var[EX][EY][EP])

Convert meteorological data from model levels to pressure levels.

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

Create meteorological data with periodic boundary conditions.

• double scan\_ctl (const char \*filename, int argc, char \*argv[], const char \*varname, int arridx, const char \*defvalue, char \*value)

Read a control parameter from file or command line.

void time2jsec (int year, int mon, int day, int hour, int min, int sec, double remain, double \*jsec)

Convert date to seconds.

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

Measure wall-clock time.

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

Write atmospheric data.

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

Write CSI data

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

Write gridded data.

- void write\_prof (const char \*filename, ctl\_t \*ctl, met\_t \*met0, met\_t \*met1, atm\_t \*atm, double t)
   Write profile data.
- void write\_station (const char \*filename, ctl\_t \*ctl, atm\_t \*atm, double t)

  Write station data.

## 5.13.1 Detailed Description

MPTRAC library declarations.

Definition in file libtrac.h.

#### 5.13.2 Function Documentation

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

Convert Cartesian coordinates to geolocation.

Definition at line 29 of file libtrac.c.

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

5.13.2.2 double deg2dx ( double dlon, double lat )

Convert degrees to horizontal distance.

Definition at line 45 of file libtrac.c.

5.13.2.3 double deg2dy ( double dlat )

Convert degrees to horizontal distance.

Definition at line 54 of file libtrac.c.

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

Convert pressure to vertical distance.

Definition at line 62 of file libtrac.c.

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

5.13.2.5 double dx2deg ( double dx, double lat )

Convert horizontal distance to degrees.

Definition at line 71 of file libtrac.c.

5.13.2.6 double dy2deg ( double dy )

Convert horizontal distance to degrees.

Definition at line 84 of file libtrac.c.

5.13.2.7 double dz2dp ( double dz, double p )

Convert vertical distance to pressure.

Definition at line 92 of file libtrac.c.

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

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

Convert geolocation to Cartesian coordinates.

Definition at line 101 of file libtrac.c.

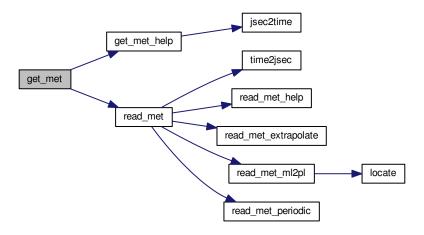
5.13.2.9 void get\_met ( ctl\_t \* ctl, char \* metbase, double t, met\_t \* met0, met\_t \* met1 )

Get meteorological data for given timestep.

Definition at line 117 of file libtrac.c.

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

Here is the call graph for this function:



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

Get meteorological data for timestep.

Definition at line 156 of file libtrac.c.

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

Here is the call graph for this function:



5.13.2.11 void intpol\_met\_2d ( double array[EX][EY], int ix, int iy, double wx, double wy, double \*var)

Linear interpolation of 2-D meteorological data.

Definition at line 182 of file libtrac.c.

```
00188
00189
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 }
```

5.13.2.12 void intpol\_met\_3d ( float array[EX][EY][EP], int ip, int ix, int iy, double wp, double wx, double wy, double \* var )

Linear interpolation of 3-D meteorological data.

Definition at line 206 of file libtrac.c.

```
00214
00215
00216
         double aux00, aux01, aux10, aux11;
00217
00218
        /* Interpolate vertically... */
        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
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 }
```

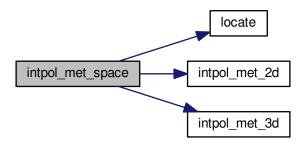
5.13.2.13 void intpol\_met\_space (  $met_t * met$ , double p, double lon, double lo

Spatial interpolation of meteorological data.

Definition at line 236 of file libtrac.c.

```
00247
00248
00249
        double wp, wx, wv;
00250
00251
        int ip, ix, iy;
00252
00253
        /* Check longitude... */
        if (met->lon[met->nx - 1] > 180 && lon < 0)
00254
00255
          lon += 360;
00256
00257
        /* Get indices... */
00258
        ip = locate(met->p, met->np, p);
00259
        ix = locate(met->lon, met->nx, lon);
00260
        iy = locate(met->lat, met->ny, lat);
00261
00262
        /* Get weights... */
        wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);
00263
00264
00265
00266
00267
        /* Interpolate... */
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);
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);
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:



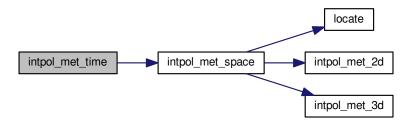
5.13.2.14 void intpol\_met\_time ( met\_t \* met0, met\_t \* met1, double ts, double p, double lon, double lat, double \* ps, double \* t, double \* u, double \* v, double \* w, double \* b2o, double \* o3 )

Temporal interpolation of meteorological data.

Definition at line 286 of file libtrac.c.

```
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:



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

Convert seconds to date.

Definition at line 341 of file libtrac.c.

```
00349
                         {
00350
       struct tm t0, *t1;
00352
00353
       time_t jsec0;
00354
00355
       t0.tm_year = 100;
        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 }
```

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

Find array index.

Definition at line 376 of file libtrac.c.

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

```
ihi = i;
00392
             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 }
```

5.13.2.17 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
          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 }
```

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

Read control parameters.

Definition at line 458 of file libtrac.c.

```
00462
00463
00464
         int ip, iq;
00465
00466
         /* Write info... */
         printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
00467
                 "(executable: %s \mid 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;
         ctl->qnt_theta = -1;
00482
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
             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...
        ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL); ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
00546
00547
         if (ctl->met_np > EP)
00548
```

```
00549
          ERRMSG("Too many levels!");
00550
        for (ip = 0; ip < ctl->met_np; ip++)
00551
           ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
00552
00553
         /* Isosurface parameters... */
00554
         ctl->isosurf
         = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00555
00556
00557
00558
         /* Diffusion parameters... */
00559
         ctl->turb_dx_trop
          = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00560
00561
        ctl->turb dx strat
            = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00562
00563
         ctl->turb_dz_trop
00564
           = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00565
         ctl->turb dz strat
00566
           = scan ctl(filename, argc, argv, "TURB DZ STRAT", -1, "0.1", NULL);
         ctl->turb_meso =
00567
00568
           scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00569
00570
        /* Life time of particles... */
        ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
ctl->tdec_strat =
00571
00572
00573
           scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00574
00575
         /\star Output of atmospheric data... \star/
        scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
00576
      atm_basename);
00577
         scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00578
         ctl->atm dt out =
00579
           scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00580
00581
        /* Output of CSI data... */
00582
        scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
00583
        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",
00584
00585
00586
                  ctl->csi_obsfile);
00587
        ctl->csi obsmin =
          scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00588
00589
        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);
00590
00591
00592
00593
         ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
        ctl->csi lon0 =
00594
        scan_ctl(filename, argc, argv, "CSI_LONO", -1, "-180", NULL);
ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
00595
00596
00597
        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);
00598
00599
00600
         ctl->csi_nv =
00601
00602
           (int) scan ctl(filename, argc, argv, "CSI NY", -1, "180", NULL);
00603
00604
        /* Output of grid data... */
00605
        scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
00606
                   ctl->grid_basename);
        scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
00607
      grid_gpfile);
00608
        ctl->grid_dt_out =
           scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
00609
          00610
         ctl->grid_sparse
00611
        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);
00612
00613
00614
        ctl->grid_nz =
00615
           (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00616
         ctl->grid_lon0 =
00617
           scan_ctl(filename, argc, argv, "GRID_LONO", -1, "-180", NULL);
00618
         ctl->grid_lon1 =
           scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
00619
         ctl->grid_nx =
00620
           (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00621
00622
         ctl->grid_lat0 =
00623
           scan_ctl(filename, argc, argv, "GRID_LATO", -1, "-90", NULL);
00624
         ctl->grid lat1 =
           scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90". NULL);
00625
00626
         ctl->grid ny =
00627
           (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
00628
00629
         /* Output of profile data... */
         scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
00630
                   ctl->prof basename);
00631
         scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
00632
```

```
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);
00633
00634
00635
         ctl->prof nz =
            (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
00636
00637
         ctl->prof lon0 =
00638
            scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
00639
00640
            scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
00641
         ctl->prof nx =
            (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00642
00643
         ctl->prof lat0 =
00644
            scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
00645
00646
            scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
         ctl->prof_ny =
00647
            (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00648
00649
00650
         /* Output of station data... */
        scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
00651
00652
                    ctl->stat_basename);
         ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
00653
00654
00655
00656 }
```

Here is the call graph for this function:



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

Read meteorological data file.

Definition at line 660 of file libtrac.c.

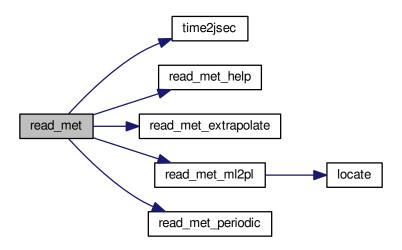
```
00663
00664
00665
        char tstr[10];
00666
00667
        static float help[EX * EY];
00668
       int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00669
00670
00671
       size_t np, nx, ny;
00672
00673
       /* Write info... */
00674
        printf("Read meteorological data: %s\n", filename);
00675
00676
        /* Get time from filename... */
        sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00677
00678
        year = atoi(tstr);
00679
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00680
        mon = atoi(tstr);
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00681
        day = atoi(tstr);
sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00682
00683
00684
        hour = atoi(tstr);
00685
        time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00686
        /* Open netCDF file... */
00687
       NC(nc_open(filename, NC_NOWRITE, &ncid));
00688
00689
00690
        /* Get dimensions... */
```

```
NC(nc_inq_dimid(ncid, "lon", &dimid));
00692
           NC(nc_inq_dimlen(ncid, dimid, &nx));
00693
           if (nx > EX)
             ERRMSG("Too many longitudes!");
00694
00695
          NC(nc_inq_dimid(ncid, "lat", &dimid));
NC(nc_inq_dimlen(ncid, dimid, &ny));
00696
00697
00698
               (ny > EY)
00699
             ERRMSG("Too many latitudes!");
00700
          NC(nc_inq_dimid(ncid, "lev", &dimid));
NC(nc_inq_dimlen(ncid, dimid, &np));
00701
00702
00703
               (np > EP)
00704
             ERRMSG("Too many levels!");
00705
00706
          /* Store dimensions... */
          met->np = (int) np;

met->nx = (int) nx;
00707
00708
          met->ny = (int) ny;
00710
          /* Get horizontal grid... */
NC(nc_inq_varid(ncid, "lon", &varid));
00711
00712
          NC(nc_get_var_double(ncid, varid, met->lon));
NC(nc_inq_varid(ncid, "lat", &varid));
NC(nc_get_var_double(ncid, varid, met->lat));
00713
00714
00715
00716
00717
           /* Read meteorological data... */
          /* Read meteorological data... */
read_met_help(ncid, "t", "T", met, met->t, 1.0);
read_met_help(ncid, "u", "U", met, met->u, 1.0);
read_met_help(ncid, "v", "V", met, met->v, 1.0);
read_met_help(ncid, "w", "W", met, met->w, 0.01f);
read_met_help(ncid, "q", "Q", met, met->help(ncid, "q", "Q", met, met->help(ncid, "o3", "o3", met, met->o3, 0.602f);
00718
00719
00720
00721
00722
00723
00724
00725
           /\star Meteo data on pressure levels... \star/
00726
          if (ctl->met_np <= 0) {</pre>
00727
00728
              /* Read pressure levels from file... */
00729
             NC(nc_inq_varid(ncid, "lev", &varid));
00730
             NC(nc_get_var_double(ncid, varid, met->p));
             for (ip = 0; ip < met->np; ip++)
met->p[ip] /= 100.;
00731
00732
00733
00734
              /* Extrapolate data for lower boundary... */
00735
             read_met_extrapolate(met);
00736
00737
00738
           /* Meteo data on model levels... */
00739
          else {
00740
00741
              /* Read pressure data from file... */
00742
             read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00743
00744
              /\star Interpolate from model levels to pressure levels... \star/
00745
             read_met_ml2pl(ctl, met, met->t);
00746
             read_met_ml2pl(ctl, met, met->u);
00747
             read_met_ml2pl(ctl, met, met->v);
00748
              read_met_ml2pl(ctl, met, met->w);
00749
              read_met_ml2pl(ctl, met, met->h2o);
00750
             read_met_ml2pl(ctl, met, met->o3);
00751
00752
             /\star Set pressure levels... \star/
             met->np = ctl->met_np;
for (ip = 0; ip < met->np; ip++)
00753
00754
00755
                met->p[ip] = ctl->met_p[ip];
00756
00757
00758
           /* Check ordering of pressure levels... */
          for (ip = 1; ip < met->np; ip++)
   if (met->p[ip - 1] < met->p[ip])
00759
00760
00761
                ERRMSG("Pressure levels must be descending!");
00762
          /* Read surface pressure... */
if (nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
   NC(nc_get_var_float(ncid, varid, help));
   for (iy = 0; iy < met->ny; iy++)
00763
00764
00765
00766
00767
                for (ix = 0; ix < met->nx; ix++)
          met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
} else if (nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00768
00769
00770
             NC(nc_get_var_float(ncid, varid, help));
00771
             for (iy = 0; iy < met->ny; iy++)
for (ix = 0; ix < met->nx; ix++)
00772
00773
                  met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00774
           } else
00775
             for (ix = 0; ix < met->nx; ix++)
                for (iy = 0; iy < met->ny; iy++)
  met->ps[ix][iy] = met->p[0];
00776
00777
```

```
00778
00779  /* Create periodic boundary conditions... */
00780  read_met_periodic(met);
00781
00782  /* Close file... */
00783  NC(nc_close(ncid));
00784 }
```

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 788 of file libtrac.c.

```
00789
                             {
00790
00791
           int ip, ip0, ix, iy;
00792
00793
           /* Loop over columns... */
00794
           for (ix = 0; ix < met->nx; ix++)
00795
             for (iy = 0; iy < met->ny; iy++) {
00796
00797
                /* Find lowest valid data point... */
for (ip0 = met->np - 1; ip0 >= 0; ip0--)
   if (!gsl_finite(met->t[ix][iy][ip0])
00798
00799
00800
                        || !gsl_finite(met->u[ix][iy][ip0])
00801
                        || !gsl_finite(met->v[ix][iy][ip0])
00802
                        || !gsl_finite(met->w[ix][iy][ip0]))
00803
                     break;
00804
00805
                /* Extrapolate... */
00806
                for (ip = ip0; ip >= 0; ip--) {
                  met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
00807
80800
00809
00810
00811
                  met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
00812
                   met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00813
00814
00815 }
```

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

Read and convert variable from meteorological data file.

Definition at line 819 of file libtrac.c.

```
00825
                        {
00826
00827
         static float help[EX * EY * EP];
00828
00829
         int ip, ix, iy, n = 0, varid;
00830
00831
          /* Check if variable exists... */
00832
          if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00833
          if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00834
00835
         /* Read data... */
NC(nc_get_var_float(ncid, varid, help));
00836
00838
00839
          /* Copy and check data... */
         for (ip = 0; ip < met->np; ip++)
  for (iy = 0; iy < met->ny; iy++)
00840
00841
              for (ix = 0; ix < met->nx; ix++) {
  dest[ix][iy][ip] = scl * help[n++];
00842
00843
                if (fabs(dest[ix][iy][ip] / scl) > 1el4)
  dest[ix][iy][ip] = GSL_NAN;
00844
00845
00846
00847 }
```

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

Convert meteorological data from model levels to pressure levels.

Definition at line 851 of file libtrac.c.

```
00854
00855
00856
        double aux[EP], p[EP], pt;
00857
00858
        int ip, ip2, ix, iy;
00859
00860
        /* Loop over columns... */
00861
        for (ix = 0; ix < met->nx; ix++)
00862
          for (iy = 0; iy < met->ny; iy++) {
00863
00864
            /* Copy pressure profile... */
for (ip = 0; ip < met->np; ip++)
00865
              p[ip] = met->pl[ix][iy][ip];
00866
00867
00868
             /* Interpolate... */
             for (ip = 0; ip < ctl->met_np; ip++) {
00869
00870
               pt = ctl->met_p[ip];
00871
               if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
                pt = p[0];
00872
00873
               else if ((pt > p[met->np - 1] && p[1] > p[0])
                 || (pt < p[met->np - 1] && p[1] < p[0]))
pt = p[met->np - 1];
00874
00875
               00876
00877
00878
00879
00880
            /* Copy data... */
for (ip = 0; ip < ctl->met_np; ip++)
    var[ix][iy][ip] = (float) aux[ip];
00881
00882
00883
00884
00885 }
```

Here is the call graph for this function:



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

Create meteorological data with periodic boundary conditions.

Definition at line 889 of file libtrac.c.

```
00890
00891
00892
        int ip, iy;
00893
        /* Check longitudes... */
00894
        if (!(fabs(met->lon[met->nx - 1] - met->lon[0])
00895
                    + met->lon[1] - met->lon[0] - 360) < 0.01))
00897
00898
00899
        /\star Increase longitude counter... \star/
        if ((++met->nx) > EX)
00900
          ERRMSG("Cannot create periodic boundary conditions!");
00901
00902
        00903
00904
      lon[0];
00905
00906
        /* Loop over latitudes and pressure levels... */
        for (iy = 0; iy < met->ny; iy++)
00907
         for (iy = 0; iy < met->ny; iy++)
for (ip = 0; ip < met->np; ip++) {
  met->ps[met->nx - 1][iy] = met->ps[0][iy];
  met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
  met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
00908
00909
00910
00911
            met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
00912
            00913
00915
00916
00917 }
```

5.13.2.24 double scan\_ctl ( const char \* filename, int argc, char \* argv[], const char \* varname, int arridx, const char \* defvalue, char \* value )

Read a control parameter from file or command line.

Definition at line 921 of file libtrac.c.

```
00928
                     {
00929
       FILE *in = NULL:
00930
00931
       char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
00932
00933
        msg[LEN], rvarname[LEN], rval[LEN];
00934
00935
       int contain = 0, i;
00936
00937
       /* Open file... */
00938
       if (filename[strlen(filename) - 1] != '-')
         if (!(in = fopen(filename, "r")))
```

```
00940
               ERRMSG("Cannot open file!");
00941
00942
          /* Set full variable name... */
          if (arridx >= 0) {
00943
           sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
00944
00945
          } else {
          sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
00947
00948
00949
00950
00951
          /* Read data... */
00952
          if (in != NULL)
          while (fgets(line, LEN, in))
if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
if (strcasecmp(rvarname, fullname1) == 0 ||
strcasecmp(rvarname, fullname2) == 0) {
00953
00954
00955
00956
00957
                    contain = 1;
00958
                    break;
00959
                 }
00960
          for (i = 1; i < argc - 1; i++)
00961
          if (strcasecmp(argv[i], fullname1) == 0 ||
               strcasecmp(argv[i], fullname2) == 0) {
sprintf(rval, "%s", argv[i + 1]);
00962
00963
00964
               contain = 1;
00965
               break;
00966
00967
00968
         /* Close file... */
         if (in != NULL)
00969
00970
            fclose(in);
00971
00972
         /* Check for missing variables... */
00973
          if (!contain) {
           if (strlen(defvalue) > 0)
   sprintf(rval, "%s", defvalue);
00974
00975
00976
            else {
00977
              sprintf(msg, "Missing variable %s!\n", fullname1);
00978
               ERRMSG(msg);
00979
00980
00981
         /* Write info... */
00982
         printf("%s = %s\n", fullname1, rval);
00983
00984
00985
          /* Return values... */
         if (value != NULL)
    sprintf(value, "%s", rval);
00986
00987
00988
         return atof(rval);
00989 }
```

5.13.2.25 void time2jsec ( int year, int mon, int day, int hour, int min, int sec, double remain, double \* jsec )

Convert date to seconds.

Definition at line 993 of file libtrac.c.

```
01001
01002
01003
       struct tm t0, t1;
01004
       t0.tm_year = 100;
01005
01006
        t0.tm_mon = 0;
01007
        t0.tm_mday = 1;
01008
        t0.tm\_hour = 0;
       t0.tm_min = 0;
t0.tm_sec = 0;
01009
01010
01011
01012
        t1.tm_year = year - 1900;
        t1.tm_mon = mon - 1;
01014
        t1.tm_mday = day;
01015
        t1.tm_hour = hour;
01016
        t1.tm_min = min;
        t1.tm_sec = sec;
01017
01018
       *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01020 }
```

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

Measure wall-clock time.

Definition at line 1024 of file libtrac.c.

```
01027
01028
01029
       static double starttime[NTIMER], runtime[NTIMER];
01030
        /* Check id...
01031
01032
        if (id < 0 || id >= NTIMER)
01033
         ERRMSG("Too many timers!");
01034
01035
        /* Start timer... */
01036
       if (mode == 1) {
        if (starttime[id] <= 0)</pre>
01038
            starttime[id] = omp_get_wtime();
01039
           ERRMSG("Timer already started!");
01040
01041
01042
01043
       /* Stop timer... */
01044
       else if (mode == 2) {
01045
        if (starttime[id] > 0) {
01046
           runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
            starttime[id] = -1;
01047
01048
         } else
01049
           ERRMSG("Timer not started!");
01050
01051
01052
        /* Print timer... */
01053
       else if (mode == 3)
         printf("%s = %g s\n", name, runtime[id]);
01054
01055 }
```

# 5.13.2.27 double tropopause ( double t, double lat )

Definition at line 1059 of file libtrac.c.

```
01061
01062
           static double doys[12]
01063
           = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01064
01065
01066
            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,
01067
01068
               -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,
01069
01070
01072
                45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
01073
               75, 77.5, 80, 82.5, 85, 87.5, 90
01074
           };
01075
01076
           static double tps[12][73]
            = { 324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6, 297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
01077
                       175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
01079
01080
                       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,
01081
01082
01083
            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,
01084
01085
01086
             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,
01087
01088
01089
01091
              284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01092
              287.5, 286.2, 285.8},
01093
            {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,
01094
01095
01096
              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,
01099
01100
            304.3, 304.9, 306, 306.6, 306.2, 306},
01101
           {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,
01102
01103
            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,
01105
01106
            148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
01107
            263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4
           205.7, 270.3, 270.6, 262.0, 260.1, 294.3, 300.4, 300.3, 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, 205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
01108
01109
01110
01111
01112
            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,
01113
01114
            273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01115
01116
            325.3, 325.8, 325.8},
           {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
            222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
01118
01119
            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,
01120
01121
01122
01123
            308.5, 312.2, 313.1, 313.3},
01124
01125
           {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,
01126
01127
01128
01129
            117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
01130
01131
            224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5
01132
            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,
01133
            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,
01134
01136
            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,
01137
01138
            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,
01139
           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,
01140
01141
            183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
01142
01143
            243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
01144
            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,
01145
01146
01147
            276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
01148
           {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
01149
01150
            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,
01151
01152
            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,
01153
            206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01155
01156
            279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
            305.1},
01157
           {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01158
           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,
01159
01160
            108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5,
01162
            102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
           109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4, 241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
01163
01164
            286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6}
01165
           {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
01166
            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,
01168
01169
            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,
01170
01171
01172
            280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
            281.7, 281.1, 281.2}
01173
01174
01175
01176
          double doy, p0, p1, pt;
01177
01178
          int imon, ilat;
          /* Get day of year... */
doy = fmod(t / 86400., 365.25);
01180
01181
01182
          while (doy < 0)
01183
             dov += 365.25;
01184
```

```
01185
         /* Get indices... */
         imon = locate(doys, 12, doy);
ilat = locate(lats, 73, lat);
01186
01187
01188
         /* Get tropopause pressure... */
01189
        p0 = LIN(lats[ilat], tps[imon][ilat],
lats[ilat + 1], tps[imon][ilat + 1], lat);
01190
01191
        p1 = LIN(lats[ilat], tps[imon + 1][ilat],
lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
01192
01193
01194
         pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01195
         /* Return tropopause pressure... */
01196
01197
         return pt;
01198 }
```

Here is the call graph for this function:



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

Write atmospheric data.

Definition at line 1202 of file libtrac.c.

```
01206
01207
01208
        FILE *in, *out;
01209
        char line[LEN];
01210
01211
01212
        double r;
01213
01214
         int ip, iq, year, mon, day, hour, min, sec;
01215
         /* Check if gnuplot output is requested... */
if (ctl->atm_gpfile[0] != '-') {
01216
01217
01218
01219
           /\star Write info... \star/
01220
           printf("Plot atmospheric data: %s.png\n", filename);
01221
           /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
01222
01223
             ERRMSG("Cannot create pipe to gnuplot!");
01224
01225
           01226
01227
01228
01229
           /\star Set time string... \star/
           jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01230
01231
01232
                    year, mon, day, hour, min);
01233
01234
           /* Dump gnuplot file to pipe... */
           if (!(in = fopen(ctl->atm_gpfile, "r")))
ERRMSG("Cannot open file!");
01235
01236
           while (fgets(line, LEN, in))
fprintf(out, "%s", line);
01237
01238
01239
           fclose(in);
01240
01241
01242
        else {
01243
01244
           /* Write info... */
```

```
printf("Write atmospheric data: %s\n", filename);
01246
01247
           /* Create file... */
          if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");
01248
01249
01250
01251
01252
        /* Write header... */
01253
        fprintf(out,
                (Out,
"# $1 = time [s]\n"
"# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01254
01255
01256
        01257
01258
01259
        fprintf(out, "\n");
01260
01261
01262
        /* Write data... */
        01263
01264
01265
          for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01266
01267
01268
01269
01270
          fprintf(out, "\n");
01271
01272
01273
        /\star Close file... \star/
01274
        fclose(out);
01275 }
```

Here is the call graph for this function:



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

Write CSI data.

Definition at line 1279 of file libtrac.c.

```
01283
01284
       static FILE *in, *out;
01286
01287
       static char line[LEN];
01288
        static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01289
01290
         rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01291
01292
        static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01293
        /* Init... */
01294
01295
        if (!init) {
01296
         init = 1;
01297
01298
          /* Check quantity index for mass... */
01299
          if (ctl->qnt_m < 0)
           ERRMSG("Need quantity mass to analyze CSI!");
01300
01301
01302
          /* Open observation data file... */
         printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
01303
01304
          if (!(in = fopen(ctl->csi_obsfile, "r")))
```

```
01305
            ERRMSG("Cannot open file!");
01306
01307
          /* Create new file... */
          printf("Write CSI data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01308
01309
            ERRMSG("Cannot create file!");
01310
01311
01312
           /* Write header... */
01313
          fprintf(out,
                   "# $1 = time [s]\n"
"# $2 = number of hits (cx)\n"
01314
01315
                   "# $3 = number of misses (cy) \n"
01316
                   "# $4 = number of false alarms (cz)\n"
01317
01318
                   "# $5 = number of observations (cx + cy)\n"
01319
                   "# $6 = number of forecasts (cx + cz)n"
01320
                   "# \$7 = bias (forecasts/observations) [ \% ] \n"
                   "# $8 = probability of detection (POD) [%%]\n" "# $9 = false alarm rate (FAR) [%%]\n" "# $10 = critical success index (CSI) [%%]\n\n");
01321
01322
01323
01324
        }
01325
01326
        /* Set time interval... */
       t0 = t - 0.5 * ctl->dt_mod;

t1 = t + 0.5 * ctl->dt_mod;
01327
01328
01329
01330
        /* Initialize grid cells... */
01331
        for (ix = 0; ix < ctl->csi_nx; ix++)
01332
         for (iy = 0; iy < ctl->csi_ny; iy++)
01333
            for (iz = 0; iz < ctl->csi_nz; iz++)
01334
              modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01335
01336
        /* Read data...
01337
        while (fgets(line, LEN, in)) {
01338
           /* Read data... *
01339
          if (sscanf(line, "%lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01340
01341
               5)
01342
             continue;
01343
01344
          /* Check time... */
01345
          <u>if</u> (rt < t0)
01346
            continue;
          if (rt > t1)
01347
01348
            break;
01349
01350
          /* Calculate indices... */
01351
          ix = (int) ((rlon - ctl->csi\_lon0))
                        / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01352
          iy = (int) ((rlat - ctl->csi_lat0)
01353
01354
                        / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
          iz = (int) ((rz - ctl - > csi_z0))
01355
01356
                       / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01357
          /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
01358
01359
              iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01360
01361
             continue:
01362
01363
           /* Get mean observation index... */
01364
          obsmean[ix][iy][iz] += robs;
01365
          obscount[ix][iy][iz]++;
01366
01367
01368
        /* Analyze model data... */
01369
        for (ip = 0; ip < atm->np; ip++) {
01370
01371
          /* Check time... */
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
01372
01373
            continue;
01374
01375
           /* Get indices... */
01376
          ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
                        / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01377
          01378
01379
          01380
01381
01382
          /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
    iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01383
01384
01385
01386
             continue;
01387
01388
           /∗ Get total mass in grid cell... ∗/
01389
          modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01390
01391
```

```
/* Analyze all grid cells... */
         for (ix = 0; ix < ctl->csi_nx; ix++)
01393
01394
           for (iy = 0; iy < ctl->csi_ny; iy++)
              for (iz = 0; iz < ctl->csi_nz; iz++) {
01395
01396
01397
                 /* Calculate mean observation index... */
                if (obscount[ix][iy][iz] > 0)
01398
01399
                   obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01400
01401
                /\star Calculate column density... \star/
                if (modmean[ix][iy][iz] > 0) {
01402
                  dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
01403
01404
01405
                   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);
01406
01407
01408
01409
                }
01410
01411
                /* Calculate CSI... */
01412
                if (obscount[ix][iy][iz] > 0) {
01413
                   if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
                       modmean[ix][iy][iz] >= ctl->csi_modmin)
01414
01415
                     cx++;
01416
                  else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
                              modmean[ix][iy][iz] < ctl->csi_modmin)
01417
01418
                     cy++;
01419
                   else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
                              modmean[ix][iy][iz] >= ctl->csi_modmin)
01420
01421
                     cz++;
01422
01423
01424
01425
         /* Write output... */
01426
         if (fmod(t, ctl->csi_dt_out) == 0) {
01427
           01428
01430
                     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);
01431
01432
01433
01434
01435
01436
            /* Set counters to zero... */
01437
           cx = cy = cz = 0;
01438
01439
         /* Close file... */
01440
01441
         if (t == ctl->t_stop)
01442
           fclose(out);
01443 }
```

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

Write gridded data.

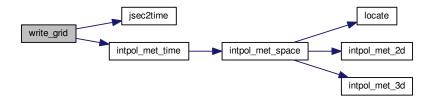
Definition at line 1447 of file libtrac.c.

```
01453
                  {
01454
01455
       FILE *in, *out;
01456
01457
       char line[LEN];
01458
       static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01459
01460
         area, rho_air, press, temp, cd, mmr, t0, t1, r;
01461
01462
       static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01463
01464
        /* Check dimensions... */
        if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
01465
         ERRMSG("Grid dimensions too large!");
01466
01467
        /\star Check quantity index for mass... \star/
01468
01469
       if (ctl->qnt_m < 0)
01470
         ERRMSG("Need quantity mass to write grid data!");
01471
01472
       /* Set time interval for output... */
01473
       t0 = t - 0.5 * ctl->dt_mod;
```

```
01474
        t1 = t + 0.5 * ctl->dt_mod;
01475
01476
         /* 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;
01477
01478
01479
01480
01481
          /* Initialize grid... */
01482
         for (ix = 0; ix < ctl->grid_nx; ix++)
            for (iy = 0; iy < ctl->grid_ny; iy++)
  for (iz = 0; iz < ctl->grid_nz; iz++)
01483
01484
01485
                 grid_m[ix][iy][iz] = 0;
01486
01487
          /* Average data... */
01488
         for (ip = 0; ip < atm->np; ip++)
01489
           if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {</pre>
01490
01491
              /* Get index... */
              ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
01492
              iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
01493
01494
              iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01495
              /* Check indices... */
if (ix < 0 || ix >= ctl->grid_nx ||
    iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01496
01497
01498
01499
                 continue:
01500
              /* Add mass... */
01501
              grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01502
01503
01504
         /* Check if gnuplot output is requested... */
if (ctl->grid_gpfile[0] != '-') {
01505
01506
01507
           /* Write info... */
printf("Plot grid data: %s.png\n", filename);
01508
01509
01510
01511
            /* Create gnuplot pipe... */
           if (!(out = popen("gnuplot", "w")))
01512
01513
            ERRMSG("Cannot create pipe to gnuplot!");
01514
           /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
01515
01516
01517
01518
            /* Set time string... */
           jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01519
01520
01521
                     year, mon, day, hour, min);
01522
01523
            /* Dump gnuplot file to pipe... */
01524
           if (!(in = fopen(ctl->grid_gpfile, "r")))
             ERRMSG("Cannot open file!");
01525
01526
            while (fgets(line, LEN, in))
01527
             fprintf(out, "%s", line);
01528
           fclose(in);
01529
         }
01530
01531
01532
01533
            /* Write info... */
           printf("Write grid data: %s\n", filename);
01534
01535
01536
            /* Create file... */
           if (!(out = fopen(filename, "w")))
01537
01538
              ERRMSG("Cannot create file!");
01539
01540
         /* Write header... */
01541
01542
         fprintf(out,
                   "# $1 = time [s] \n"
01544
                   "# $2 = altitude [km] \n"
                   "# $3 = longitude [deg] \n"
01545
                   "# $4 = latitude [deg]\n"
01546
                   "# $5 = surface area [km^2]\n"
"# $6 = layer width [km]\n"
01547
01548
01549
                   "# $7 = temperature [K] \n"
01550
                   "# $8 = \text{column density } [kg/m^2] \n"
                   "# $9 = mass mixing ratio [1]\n\n");
01551
01552
01553
         /* Write data... */
         for (ix = 0; ix < ctl->grid_nx; ix++) {
01554
          if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01556
              fprintf(out, "\n");
01557
            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++)
01558
01559
01560
```

```
if (!ctl->grid_sparse
01562
                        || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01563
                     /* Set coordinates... */
01564
                     z = ctl->grid_z0 + dz * (iz + 0.5);
lon = ctl->grid_lon0 + dlon * (ix + 0.5);
01565
01566
                     lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01567
01568
01569
                     /\star Get pressure and temperature... \star/
01570
                     press = P(z);
                     intpol_met_time(met0, met1, t, press, lon, lat,
    NULL, &temp, NULL, NULL, NULL, NULL, NULL);
01571
01572
01573
01574
                      /* Calculate surface area... */
                     area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
 * cos(lat * M_PI / 180.);
01575
01576
01577
01578
                     /* Calculate column density... */
cd = grid_m[ix][iy][iz] / (1e6 * area);
01579
01580
                     /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01581
01582
01583
01584
01585
                      /* Write output... */
01586
                     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g\n", t, z, lon, lat, area, dz, temp, cd, mmr);
01587
01588
01589
             }
01590
          }
01591
01592
           /* Close file... */
01593
          fclose(out);
01594 }
```

Here is the call graph for this function:



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

Write profile data.

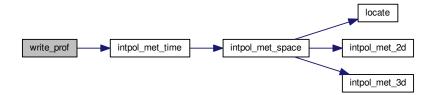
Definition at line 1598 of file libtrac.c.

```
01604
                    {
01605
01606
        static FILE *in, *out;
01607
01608
        static char line[LEN];
01609
        static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
01610
          rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, mmr, h2o, o3;
01611
01612
01613
01614
        static int init, obscount[GX][GY], ip, ix, iy, iz;
01615
         /* Init... */
01616
        if (!init) {
01617
01618
          init = 1;
01619
01620
          /* Check quantity index for mass... */
```

```
if (ctl->qnt_m < 0)
             ERRMSG("Need quantity mass!");
01622
01623
           /* Check dimensions... */
01624
           if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
    ERRMSG("Grid dimensions too large!");
01625
01626
01627
01628
           /* Open observation data file... */
01629
           printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01630
            if (!(in = fopen(ctl->prof_obsfile, "r")))
             ERRMSG("Cannot open file!");
01631
01632
           /* Create new file... */
01633
           printf("Write profile data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01634
01635
             ERRMSG("Cannot create file!");
01636
01637
01638
           /* Write header... */
01639
           fprintf(out,
                    "# $1
                            = time [s]\n"
01641
                    "# $2
                            = altitude [km]\n"
                            = longitude [deg]\n"
01642
                     "# $3
                    "# $4 = latitude [deg]\n"
01643
                     "# $5 = pressure [hPa]\n"
01644
01645
                    "# $6 = temperature [K]\n"
                    "# $7 = mass mixing ratio [1]\n"
01646
01647
                    "# $8 = H20 volume mixing ratio [1]\n"
01648
                    "# $9 = 03 volume mixing ratio [1]\n"
                     "# $10 = mean BT index [K] \n");
01649
01650
01651
           /* Set grid box size... */
01652
           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;
01653
01654
01655
01656
         /* Set time interval... */
01657
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01658
01660
01661
         /* Initialize... */
         for (ix = 0; ix < ctl->prof_nx; ix++)
01662
          for (iy = 0; iy < ctl->prof_ny; iy++) {
01663
             obsmean[ix][iy] = 0;
01664
              obscount[ix][iy] = 0;
01665
01666
              tmean[ix][iy] = 0;
01667
              for (iz = 0; iz < ctl->prof_nz; iz++)
01668
               mass[ix][iy][iz] = 0;
01669
01670
01671
         /* Read data... */
01672
         while (fgets(line, LEN, in)) {
01673
          /* Read data... */
if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01674
01675
01676
             continue;
01677
01678
           /* Check time... */
01679
           if (rt < t0)</pre>
01680
             continue;
           if (rt > t1)
01681
01682
            break;
01683
           /* Calculate indices... */
          ix = (int) ((rlon - ctl->prof_lon0) / dlon);
iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01685
01686
01687
           /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01688
01689
             continue;
01691
01692
           /\star Get mean observation index... \star/
           obsmean[ix][iy] += robs;
tmean[ix][iy] += rt;
01693
01694
           obscount[ix][iy]++;
01695
01696
01697
01698
         /* Analyze model data... */
01699
         for (ip = 0; ip < atm->np; ip++) {
01700
01701
           /* Check time... */
01702
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
01703
             continue;
01704
           /* Get indices... */
01705
           ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
01706
01707
```

```
iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01709
01710
            /* Check indices... */
           if (ix < 0 || ix >= ctl->prof_nx ||
    iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01711
01712
01713
              continue;
01714
01715
            /\star Get total mass in grid cell... \star/
01716
           mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01717
01718
01719
         /* Extract profiles... */
         for (ix = 0; ix < ctl->prof_nx; ix++)
  for (iy = 0; iy < ctl->prof_ny; iy++)
01720
01721
01722
              if (obscount[ix][iy] > 0) {
01723
                /* Write output... */
fprintf(out, "\n");
01724
01725
01726
01727
                 /* Loop over altitudes... */
01728
                 for (iz = 0; iz < ctl->prof_nz; iz++) {
01729
01730
                   /* Set coordinates... */
                   z = ctl->prof_z0 + dz * (iz + 0.5);
lon = ctl->prof_lon0 + dlon * (ix + 0.5);
01731
01732
01733
                   lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01734
01735
                   /\star Get meteorological data... \star/
01736
                   press = P(z);
                   01737
01738
01739
                   /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
01740
01741
                  area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
 * cos(lat * M_PI / 180.);
01742
01743
01744
                   mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
01745
01746
                   /* Write output... */
                   fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g \n", tmean[ix][iy] / obscount[ix][iy],
01747
01748
                             z, lon, lat, press, temp, mmr, h2o, o3,
obsmean[ix][iy] / obscount[ix][iy]);
01749
01750
01751
01752
01753
01754
         /* Close file... */
         if (t == ctl->t_stop)
01755
01756
            fclose(out):
01757 }
```

Here is the call graph for this function:



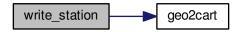
5.13.2.32 void write\_station ( const char \* filename, ctl\_t \* ctl, atm\_t \* atm, double t)

Write station data.

Definition at line 1761 of file libtrac.c.

```
01765
                    {
01766
01767
        static FILE *out;
01768
01769
        static double rmax2, t0, t1, x0[3], x1[3];
01770
01771
        static int init, ip, iq;
01772
01773
        /* Init... */
01774
        if (!init) {
01775
          init = 1;
01776
01777
          /* Write info... */
01778
          printf("Write station data: %s\n", filename);
01779
01780
           /\star Create new file... \star/
          if (!(out = fopen(filename, "w")))
01781
            ERRMSG("Cannot create file!");
01782
01783
01784
           /* Write header... */
01785
          fprintf(out,
                   "# $1 = time [s]\n"
"# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01786
01787
01788
          01789
01790
01791
01792
01793
          /\star Set geolocation and search radius... \star/
01794
01795
          geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
01796
          rmax2 = gsl_pow_2(ctl->stat_r);
01797
01798
        /* Set time interval for output... */
t0 = t - 0.5 * ctl->dt_mod;
01799
01800
        t1 = t + 0.5 * ct1 -> dt_mod;
01801
01803
         /* Loop over air parcels... */
01804
        for (ip = 0; ip < atm->np; ip++) {
01805
01806
           /* Check time... */
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
01807
01808
            continue;
01809
01810
           /\star Check station flag... \star/
01811
          if (ctl->qnt_stat >= 0)
01812
            if (atm->q[ctl->qnt_stat][ip])
01813
               continue:
01814
01815
           /* Get Cartesian coordinates... */
01816
          geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
01817
01818
           /\star Check horizontal distance... \star/
          if (DIST2(x0, x1) > rmax2)
01819
01820
            continue;
01822
           /* Set station flag... */
01823
          if (ctl->qnt_stat >= 0)
01824
            atm->q[ctl->qnt_stat][ip] = 1;
01825
          /* Write data... */
fprintf(out, "%.2f %g %g %g",
01826
01827
01828
                  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]);
01829
01830
01831
01832
01833
          fprintf(out, "\n");
01835
01836
        /* Close file... */
        if (t == ctl->t_stop)
01837
01838
          fclose(out);
01839 }
```

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)
      if((ptr=calloc((size_t)(n), sizeof(type))) == NULL)
    ERRMSG("Out of memory!");
00055
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) {
00081
        if((cmd)!=NC_NOERR)
00082
             ERRMSG(nc_strerror(cmd));
00083
00084
```

5.14 libtrac.h 117

```
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 /* -----
00132
         Dimensions...
00133
00134
00136 #define LEN 5000
00137
00139 #define NP 10000000
00140
00142 #define NQ 5
00143
00145 #define EP 73
00146
00148 #define EX 721
00149
00151 #define EY 361
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 ant v:
```

```
00207
00209
        int qnt_w;
00210
00212
        int qnt_h2o;
00213
00215
        int ant 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;
        double dt_mod;
00233
00234
00236
        double dt_met;
00237
00239
        int met_np;
00240
00242
        double met_p[EP];
00243
00246
        int isosurf;
00247
00249
        char balloon[LEN];
00250
00252
        double turb_dx_trop;
00253
        double turb_dx_strat;
00256
00258
        double turb_dz_trop;
00259
        double turb_dz_strat;
00261
00262
        double turb_meso;
00265
00267
        double tdec_trop;
00268
        double tdec_strat;
00270
00271
00273
        char atm_basename[LEN];
00274
00276
        char atm_gpfile[LEN];
00277
00279
        double atm_dt_out;
00280
00282
        char csi_basename[LEN];
00283
00285
        double csi_dt_out;
00286
00288
        char csi_obsfile[LEN];
00289
00291
        double csi obsmin;
00292
00294
        double csi_modmin;
00295
00297
        int csi_nz;
00298
00300
        double csi_z0;
00301
00303
        double csi_z1;
00304
00306
        int csi_nx;
00307
        double csi_lon0;
00309
00310
        double csi_lon1;
00313
00315
        int csi_ny;
00316
00318
        double csi_lat0;
00319
        double csi_lat1;
00322
00324
        char grid_basename[LEN];
00325
00327
        char grid_gpfile[LEN];
00328
00330
        double grid_dt_out;
00331
00333
        int grid_sparse;
00334
00336
        int grid_nz;
```

5.14 libtrac.h 119

```
00339
        double grid_z0;
00340
00342
        double grid_z1;
00343
        int grid_nx;
00345
00346
        double grid_lon0;
00349
00351
        double grid_lon1;
00352
        int grid_ny;
00354
00355
00357
        double grid_lat0;
00358
00360
        double grid_lat1;
00361
        char prof_basename[LEN];
00363
00364
00366
        char prof_obsfile[LEN];
00367
00369
        int prof_nz;
00370
00372
        double prof_z0;
00373
00375
        double prof_z1;
00376
00378
        int prof_nx;
00379
00381
        double prof_lon0;
00382
00384
        double prof_lon1;
00385
00387
        int prof_ny;
00388
00390
        double prof_lat0;
00391
00393
        double prof_lat1;
00394
00396
        char stat_basename[LEN];
00397
00399
        double stat_lon;
00400
00402
        double stat lat;
00403
00405
        double stat_r;
00406
00407 } ctl_t;
00408
00410 typedef struct {
00411
        int np;
00414
00416
        double time[NP];
00417
        double p[NP];
00419
00420
        double lon[NP];
00423
00425
        double lat[NP];
00426
00428
        double q[NQ][NP];
00429
00431
        double up[NP];
00432
00434
        double vp[NP];
00435
00437
        double wp[NP];
00438
00439 } atm_t;
00440
00442 typedef struct {
00443
00445
        double time;
00446
00448
        int nx;
00449
00451
        int ny;
00452
00454
        int np;
00455
00457
        double lon[EX];
00458
00460
        double lat[EY];
00461
00463
        double p[EP];
00464
00466
        double ps[EX][EY];
```

```
00467
00469
        float pl[EX][EY][EP];
00470
00472
       float t[EX][EY][EP];
00473
00475
        float u[EX][EY][EP];
00476
00478
        float v[EX][EY][EP];
00479
       float w[EX][EY][EP];
00481
00482
00484
       float h2o[EX][EY][EP];
00485
00487
       float o3[EX][EY][EP];
00488
00489 } met_t;
00490
00491 /* -
00492
        Functions...
00493
00494
00496 void cart2geo(
00497
       double *x,
00498
       double *z,
00499
        double *lon,
00500
       double *lat);
00501
00503 double deg2dx(
00504
       double dlon,
00505
       double lat);
00506
00508 double deg2dy(
00509
       double dlat);
00510
00512 double dp2dz(
       double dp,
00513
00514
       double p);
00515
00517 double dx2deg(
00518
       double dx,
00519
       double lat);
00520
00522 double dy2deg(
       double dy);
00523
00524
00526 double dz2dp(
00527
       double dz,
00528
       double p);
00529
00531 void geo2cart(
00532
       double z,
00533
       double lon,
00534
       double lat,
00535
       double *x);
00536
00538 void get_met(
       ctl_t * ctl,
char *metbase,
00540
00541
        double t,
       met_t * met0,
met_t * met1);
00542
00543
00544
00546 void get_met_help(
00547
       double t,
00548
        int direct,
00549
       char *metbase,
00550
       double dt_met,
       char *filename);
00551
00552
00554 void intpol_met_2d(
00555
       double array[EX][EY],
00556
        int ix,
00557
        int iy,
00558
       double wx,
       double wy,
00559
00560
       double *var);
00561
00563 void intpol_met_3d(
       float array[EX][EY][EP],
00564
00565
        int ip,
00566
        int ix,
00567
        int iy,
00568
        double wp,
00569
        double wx,
00570
        double wy,
00571
        double *var);
00572
```

5.14 libtrac.h 121

```
00574 void intpol_met_space(
00575
       met_t * met,
        double p,
00576
00577
        double lon,
00578
        double lat,
00579
        double *ps,
00580
        double *t,
00581
        double *u,
00582
        double *v,
00583
        double *w,
00584
        double *h2o,
00585
        double *o3);
00586
00588 void intpol_met_time(
       met_t * met0,
met_t * met1,
00589
00590
00591
        double ts,
00592
        double p, double lon,
00593
00594
        double lat,
00595
        double *ps,
00596
        double *t,
00597
        double *u,
00598
        double *v,
00599
        double *w,
00600
        double *h2o,
00601
        double *o3);
00602
00604 void jsec2time(
00605
        double jsec,
00606
        int *vear.
00607
        int *mon,
00608
        int *day,
00609
        int *hour,
00610
        int *min,
00611
        int *sec,
00612
        double *remain);
00613
00615 int locate(
00616
      double *xx,
00617
        int n,
       double x);
00618
00619
00621 void read_atm(
00622 const char *filename,
00623
        ctl_t * ctl,
00624
       atm_t * atm);
00625
00627 void read ctl(
00628 const char *filename,
00629
        int argc,
00630
       char *argv[],
00631
        ctl_t * ctl);
00632
00634 void read_met(
       ctl_t * ctl,
char *filename,
00635
00637
        met_t * met);
00638
00640 void read_met_extrapolate(
00641
       met_t * met);
00642
00644 void read_met_help(
00645
      int ncid,
00646
        char *varname,
00647
        char *varname2,
00648
        met_t * met,
float dest[EX][EY][EP],
00649
00650
        float scl);
00651
00653 void read_met_ml2pl(
       ctl_t * ctl,
met_t * met,
00654
00655
        float var[EX][EY][EP]);
00656
00657
00659 void read_met_periodic(
00660
       met_t * met);
00661
00663 double scan_ctl(
00664
        const char *filename,
00665
        int argc,
        char *argv[],
const char *varname,
00666
00667
00668
        int arridx,
        const char *defvalue,
00669
00670
        char *value);
00671
```

```
00673 void time2jsec(
00674
        int year,
00675
        int mon,
00676
        int day,
00677
        int hour,
00678
        int min.
00679
        int sec,
00680
        double remain,
00681
        double *jsec);
00682
00684 void timer(
00685 const char *name,
        int id,
00686
00687
        int mode);
00688
00689 /* Get tropopause pressure... */
00690 double tropopause(
00691
        double t,
00692
        double lat);
00693
00695 void write_atm(
00696 const char *filename,
        ctl_t * ctl,
atm_t * atm,
double t);
00697
00698
00699
00700
00702 void write_csi(
00703 const char *filename,
        ctl_t * ctl,
atm_t * atm,
00704
00705
00706
        double t);
00707
00709 void write_grid(
00710
        const char *filename,
        ctl_t * ctl,

met_t * met0,

met_t * met1,

atm_t * atm,
00711
00712
00713
00714
00715
        double t);
00716
00718 void write_prof(
00719 const char *filename,
00720
        ctl_t * ctl,
met_t * met0,
00721
00722
        met_t * met1,
00723
        atm_t * atm,
00724
        double t);
00725
00727 void write_station(
00728 const char *filename,
        ctl_t * ctl,
atm_t * atm,
00730
00731
        double t);
```

# 5.15 match.c File Reference

Calculate deviations between two trajectories.

# Functions

• int main (int argc, char \*argv[])

## 5.15.1 Detailed Description

Calculate deviations between two trajectories.

Definition in file match.c.

#### 5.15.2 Function Documentation

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

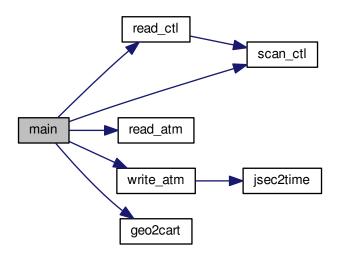
Definition at line 28 of file match.c.

```
00030
00031
00032
        ctl_t ctl;
00033
00034
        atm t *atm1, *atm2, *atm3;
00035
00036
        FILE *out;
00037
00038
        char filename[LEN];
00039
        double filter_dt, x1[3], x2[3], dh, dq[NQ], dv, lh = 0, lt = 0, lv = 0;
00040
00041
00042
        int filter, ip1, ip2, iq, n;
00043
00044
        /* Allocate... */
00045
        ALLOC(atm1, atm_t, 1);
00046
        ALLOC(atm2, atm_t, 1);
00047
        ALLOC(atm3, atm t, 1);
00048
00049
        /* Check arguments... */
00050
        if (argc < 5)</pre>
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);
00054
00055
00056
        filter_dt = scan_ctl(argv[1], argc, argv, "FILTER_DT", -1, "0", NULL);
00057
00058
        /* Read atmospheric data... */
00059
        read_atm(argv[2], &ctl, atm1);
00060
        read_atm(argv[3], &ctl, atm2);
00061
00062
        /* Write info... */
00063
        printf("Write transport deviations: %s\n", argv[4]);
00064
00065
        /* Create output file... */
        if (!(out = fopen(argv[4], "w")))
00066
          ERRMSG("Cannot create file!");
00067
00068
00069
        /* Write header... */
       00070
00071
00072
                 "# $2 = altitude [km] \n"
                 "# \$3 = longitude [deg] \n" "# <math>\$4 = latitude [deg] \n");
00073
        for (iq = 0; iq < ctl.nq; iq++)
fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00074
00075
00076
                  ctl.qnt_unit[iq]);
        00077
00078
00079
00080
                 "# \$d = horizontal length of trajectory [km]\n"
00081
                 "# \$%d = vertical deviation [km]\n"
                 "# \$%d = horizontal deviation [km]\n",
00082
00083
                5 + ctl.nq, 6 + ctl.nq, 7 + ctl.nq, 8 + ctl.nq, 9 + ctl.nq);
        for (iq = 0; iq < ctl.nq; iq+)
    fprintf(out, "# $%d = %s deviation [%s]\n", ctl.nq + iq + 10,</pre>
00084
00085
        ctl.qnt_name[iq], ctl.qnt_unit[iq]);
fprintf(out, "\n");
00086
00087
00088
00089
        /\star Filtering of reference time series... \star/
00090
        if (filter) {
00091
00092
          /* Copy data... */
00093
          memcpy(atm3, atm2, sizeof(atm_t));
00094
00095
          /* Loop over data points... */
00096
          for (ip1 = 0; ip1 < atm2->np; ip1++) {
00097
           n = 0;
00098
            atm2->p[ip1] = 0;
00099
            for (iq = 0; iq < ctl.nq; iq++)</pre>
00100
              atm2->q[iq][ip1] = 0;
             for (ip2 = 0; ip2 < atm2->np; ip2++)
00101
              if (fabs(atm2->time[ip1] - atm2->time[ip2]) < filter_dt) {
  atm2->p[ip1] += atm3->p[ip2];
00102
00103
00104
                for (iq = 0; iq < ctl.nq; iq++)</pre>
00105
                  atm2->q[iq][ip1] += atm3->q[iq][ip2];
```

```
00106
                 n++;
00107
00108
             atm2->p[ip1] /= n;
             for (iq = 0; iq < ctl.nq; iq++)</pre>
00109
00110
               atm2->q[iq][ip1] /= n;
00111
00112
00113
           /\star Write filtered data... \star/
00114
           sprintf(filename, "%s.filt", argv[3]);
00115
           write_atm(filename, &ctl, atm2, 0);
00116
00117
00118
         /* Loop over air parcels (reference data)... */
00119
        for (ip2 = 0; ip2 < atm2->np; ip2++) {
00120
00121
           /* Get trajectory length... */
00122
           if (ip2 > 0) {
             geo2cart(0, atm2->lon[ip2 - 1], atm2->lat[ip2 - 1], x1);
geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00123
00125
             lh += DIST(x1, x2);
             lv += fabs(Z(atm2->p[ip2 - 1]) - Z(atm2->p[ip2]));
lt = fabs(atm2->time[ip2] - atm2->time[0]);
00126
00127
00128
           }
00129
00130
           /* Init... */
00131
           n = 0;
00132
           dh = 0;
00133
           dv = 0;
           for (iq = 0; iq < ctl.nq; iq++)</pre>
00134
            dq[iq] = 0;
00135
00136
           geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00137
00138
           /\star Find corresponding time step (test data)... \star/
00139
           for (ip1 = 0; ip1 < atm1->np; ip1++)
             00140
00141
00142
               /* Calculate deviations... */
00144
               geo2cart(0, atm1->lon[ip1], atm1->lat[ip1], x1);
00145
                dh += DIST(x1, x2);
00146
               dv += Z(atm1->p[ip1]) - Z(atm2->p[ip2]);
               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... */
00153
           if (n > 0) {
             fprintf(out, "%.2f %.4f %.4f %.4f",
00154
                     atm2->time[ip2], Z(atm2->p[ip2]),
atm2->lon[ip2], atm2->lat[ip2]);
00155
00156
             for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");</pre>
00157
00158
00159
               fprintf(out, ctl.qnt_format[iq], atm2->q[iq][ip2]);
00160
             fprintf(out, " %.2f %g %g %g %g", lt, lv, lh, dv / n, dh / n);
00161
             for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");</pre>
00163
00164
               fprintf(out, ctl.qnt_format[iq], dq[iq] / n);
00165
             fprintf(out, "\n");
00166
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 }
```

5.16 match.c 125

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
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
         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
         ALLOC(atm1, atm_t, 1);
         ALLOC(atm2, atm_t, 1);
ALLOC(atm3, atm_t, 1);
00046
00047
00048
         /* Check arguments... */
```

```
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"
00072
                  "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00073
        for (iq = 0; iq < ctl.nq; iq++)
    fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],</pre>
00074
00075
                    ctl.qnt_unit[iq]);
00076
        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
         /* Filtering of reference time series... */
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
               atm2->q[iq][ip1] /= n;
00110
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... */
        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[] )

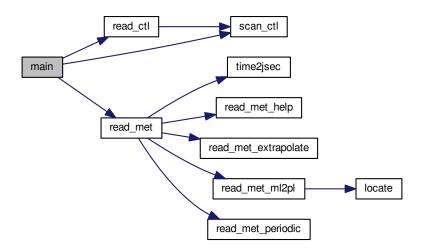
Definition at line 27 of file met\_map.c.

```
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... */
if (!(in = fopen(argv[i], "r")))
00056
00057
00058
            continue:
00059
           else
00060
            fclose(in);
00061
           read_met(&ctl, argv[i], met);
00062
00063
           /* Find nearest pressure level... */
           for (ip2 = 0; ip2 < met->np; ip2++) {
  dz = fabs(Z(met->p[ip2]) - z);
00064
00065
             if (dz < dzmin) {
               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
               vm[ix][iy] += met->v[ix][iy][ip];
00078
                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... */
00088
        printf("Write meteorological data file: %s\n", argv[2]);
         if (!(out = fopen(argv[2], "w")))
00089
           ERRMSG("Cannot create file!");
00090
00091
00092
         /* Write header... */
00093
         fprintf(out,
00094
                 "# $1 = time [s] \n"
                  "# $2
00095
                        = altitude [km] \n"
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
```

5.18 met map.c 129

```
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 >Initial, met >Initial; met >Initia
 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

```
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
the Free Software Foundation, either version 3 of the License, or
00005
00006
00007
           (at your option) any later version.
00008
00009
           MPTRAC is distributed in the hope that it will be useful,
          but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
00012
          GNU General Public License for more details.
00013
          You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
00016
00017
          Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
```

```
00027 int main(
00028
        int argc,
00029
        char *argv[]) {
00030
00031
        ctl t ctl;
00032
00033
        met_t *met;
00034
00035
        FILE *in, *out;
00036
        static double dz, dzmin = 1e10, z, timem[EX][EY], psm[EX][EY], tm[EX][EY],
um[EX][EY], vm[EX][EY], wm[EX][EY], h2om[EX][EY], o3m[EX][EY];
00037
00038
00039
00040
        static int i, ip, ip2, ix, iy, np[EX][EY];
00041
         /* Allocate... */
00042
00043
        ALLOC(met, met_t, 1);
00044
         /\star Check arguments... \star/
00045
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
         /* Loop over files... ∗/
00053
00054
         for (i = 3; i < argc; i++) {</pre>
00055
00056
           /* Read meteorological data... */
           if (!(in = fopen(argv[i], "r")))
00057
00058
             continue;
00059
00060
             fclose(in);
00061
           read_met(&ctl, 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
00066
             if (dz < dzmin) {
00067
                dzmin = dz;
00068
               ip = ip2;
00069
             }
00070
           }
00071
00072
           /* Average data... */
00073
           for (ix = 0; ix < met->nx; ix++)
             for (iy = 0; iy < met->ny; iy++) {
  timem[ix][iy] += met->time;
00074
00075
00076
                tm[ix][iy] += met->t[ix][iy][ip];
                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];
00080
00081
                o3m[ix][iy] += met->o3[ix][iy][ip];
               psm[ix][iy] += met >05[ix][iy];
psm[ix][iy] += met ->ps[ix][iy];
np[ix][iy]++;
00082
00083
00084
00085
00086
00087
        /* Create output file... */
        printf("Write meteorological data file: %s\n", argv[2]);
if (!(out = fopen(argv[2], "w")))
00088
00089
00090
           ERRMSG("Cannot create file!");
00091
         /* Write header... */
00092
00093
        00094
                         = time [s]\n"
                  "# $2 = altitude [km] \n"
00095
                         = longitude [deg]\n"
00096
00097
                  "# $4 = latitude [deg]\n"
                  "# $5 = pressure [hPa] n"
00098
                  "# $6 = temperature [K]\n"
"# $7 = zonal wind [m/s]\n"
00099
00100
                  "# $8 = meridional wind [m/s]\n'
00101
                  "# $9 = vertical wind [hPa/s]\n"
00102
00103
                  "# $10 = H20 volume mixing ratio [1]\n"
                  "# $11 = 03 volume mixing ratio [1]\n"
"# $12 = surface pressure [hPa]\n");
00104
00105
00106
         /* Write data... */
00107
        for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
00108
00109
00110
           for (ix = 0; ix < met->nx; ix++)
             00111
00112
00113
```

```
met->lon[ix] - 360.0, met->lat[iy], met->p[ip],
                               let->loft[ix] - 300.0, inet->lat[iy], inet->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]);
00116
00117
00118
              for (ix = 0; ix < met->nx; ix++)
00119
                00120
00121
00122
                              timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
00123
                                met \rightarrow lon[ix], met \rightarrow lat[iy], met \rightarrow p[ip],
                               tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy], 
vm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy], 
h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00124
00125
00126
00127
                               psm[ix][iy] / np[ix][iy]);
00128
00129
           /* Close file... */
00130
00131
           fclose(out);
00132
00133
            /* Free... */
           free (met);
00134
00135
00136
           return EXIT_SUCCESS;
00137 }
```

## 5.19 met prof.c File Reference

Extract vertical profile from meteorological data.

#### **Functions**

• int main (int argc, char \*argv[])

#### 5.19.1 Detailed Description

Extract vertical profile from meteorological data.

Definition in file met\_prof.c.

## 5.19.2 Function Documentation

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

Definition at line 38 of file met prof.c.

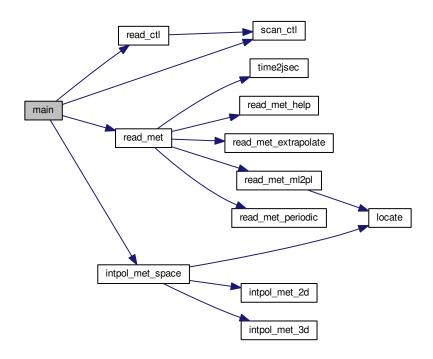
```
00041
00042
        ctl_t ctl;
00043
00044
        met t *met;
00045
00046
        FILE *in, *out;
00047
        static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ], lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ],
00048
00049
00050
         u, um[NZ], v, vm[NZ], w, wm[NZ], h2o, h2om[NZ], o3, o3m[NZ];
00051
00052
        static int i, iz, np[NZ];
00053
00054
        /* Allocate... */
00055
        ALLOC(met, met_t, 1);
00056
00057
        /* Check arguments... */
00058
        if (argc < 4)
00059
          ERRMSG("Give parameters: <ctl>   <met0> [ <met1> ... ]");
```

```
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);
dlon1 = scan_ctl(argv[1], argc, argv, "LON1", -1, "0", NULL);
dlon2 = 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
00065
00067
00068
00069
00070
00071
00072
00073
            /* Loop over input files... */
00074
            for (i = 3; i < argc; i++) {</pre>
00075
              /* Read meteorological data... */
if (!(in = fopen(argv[i], "r")))
00076
00077
                 continue;
00079
00080
                 fclose(in);
00081
               read_met(&ctl, argv[i], met);
00082
00083
               /* Average... */
for (z = z0; z <= z1; z += dz) {
00084
                iz = (int) ((z - z0) / dz);
00085
00086
                  if (iz < 0 || iz > NZ)
00087
                    ERRMSG("Too many altitudes!");
00088
                  for (lon = lon0; lon <= lon1; lon += dlon)</pre>
                    00089
00090
00091
00092
00093
                              && gsl_finite(v) && gsl_finite(w)) {
                          timem[iz] += met->time;
lonm[iz] += lon;
latm[iz] += lat;
00094
00095
00096
                          tm[iz] += t;
00098
                          um[iz] += u;
00099
                           vm[iz] += v;
                           wm[iz] += w;
00100
                          h2om[iz] += h2o;
o3m[iz] += o3;
00101
00102
00103
                          np[iz]++;
00104
00105
                    }
00106
             }
00107
           }
00108
           /* Normalize... */
00109
           for (z = z0; z \le z1; z += dz) {
00110
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
00115
                 tm[iz] /= np[iz];
00116
00117
                 um[iz] /= np[iz];
                 vm[iz] /= np[iz];
wm[iz] /= np[iz];
h2om[iz] /= np[iz];
00118
00119
00120
                 o3m[iz] /= np[iz];
00121
00122
              } else {
               timem[iz] = GSL_NAN;
00123
00124
                 lonm[iz] = GSL_NAN;
                 latm[iz] = GSL_NAN;
00125
00126
                 tm[iz] = GSL_NAN;
um[iz] = GSL_NAN;
00127
00128
                 vm[iz] = GSL_NAN;
                  wm[iz] = GSL_NAN;
00129
                 h2om[iz] = GSL_NAN;
o3m[iz] = GSL_NAN;
00130
00131
00132
00133
00134
00135
            /* Create output file... */
00136
           printf("Write meteorological data file: %s\n", argv[2]);
00137
           if (!(out = fopen(argv[2], "w")))
              ERRMSG("Cannot create file!");
00138
00139
           /* Write header... */
00140
00141
           fprintf(out,
00142
                        "# $1 = time [s] \n"
                        "# $2 = altitude [km]\n"
00143
                       "# $3 = longitude [deg]\n"
"# $4 = latitude [deg]\n"
"# $5 = pressure [hPa]\n"
00144
00145
00146
```

5.20 met prof.c 133

```
"# $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 = H2O volume mixing ratio [1]\n"
"# $11 = O3 volume mixing ratio [1]\n\n");
00151
00152
00153
00154
        /* Write data... */
       00155
00156
00157
00158
00159
00160
00161
00162
        /* Close file... */
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 /*
           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
           GNU General Public License for more details.
```

```
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"
00026
00027 /* -----
            Dimensions...
00028
00029
00030
00031 /* Maximum number of altitudes. */
00032 #define NZ 1000
00033
00034 /*
00035
            Main...
00037
00038 int main(
00039
           int argc,
00040
          char *argv[]) {
00041
00042
          ctl_t ctl;
00043
00044
           met_t *met;
00045
00046
          FILE *in, *out;
00047
           static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ],
u, um[NZ], v, vm[NZ], w, wm[NZ], h2o, h2om[NZ], o3, o3m[NZ];
00048
00049
00050
00051
00052
           static int i, iz, np[NZ];
00053
00054
           /* Allocate... */
00055
           ALLOC(met, met_t, 1);
00056
00057
            /* Check arguments... */
00058
           if (argc < 4)
00059
             ERRMSG("Give parameters: <ctl>   <met0> [ <met1> ... ]");
00060
00061
           /* Read control parameters... */
          /* Read control parameters... */
read_ctl(argv[1], argc, argv, &ctl);
z0 = scan_ctl(argv[1], argc, argv, "Z0", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "Z1", -1, "60", NULL);
dz = scan_ctl(argv[1], argc, argv, "DZ", -1, "1", NULL);
lon0 = scan_ctl(argv[1], argc, argv, "LON0", -1, "0", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "LON1", -1, "0", NULL);
00062
00063
00064
00065
00066
00067
           ion1 = 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);
00068
00069
00070
00071
00072
00073
            /* Loop over input files... */
00074
           for (i = 3; i < argc; i++) {
00075
              /* Read meteorological data... */
if (!(in = fopen(argv[i], "r")))
00076
00077
00078
                continue;
00079
              else
08000
                fclose(in);
00081
              read_met(&ctl, argv[i], met);
00082
               /* Average... */
00083
00084
              for (z = z0; z \le z1; z += dz) {
                 iz = (int) ((z - z0) / dz);
if (iz < 0 || iz > NZ)
00085
00086
                    ERRMSG("Too many altitudes!");
                 for (lon = lon0; lon <= lon1; lon += dlon)
  for (lat = lat0; lat <= lat1; lat += dlat) {</pre>
00088
00089
                       intpol_met_space(met, P(z), lon, lat, NULL,
00090
                       &t, &u, &v, &w, &h2o, &o3); if (gsl_finite(t) && gsl_finite(u)
00091
00092
00093
                             && gsl_finite(v) && gsl_finite(w)) {
00094
                          timem[iz] += met->time;
                          lonm[iz] += lon;
latm[iz] += lat;
00095
00096
                          tm[iz] += t;
00097
                          um[iz] += u;
00098
00099
                          vm[iz] += v;
00100
                          wm[iz] += w;
00101
                          h2om[iz] += h2o;
00102
                          o3m[iz] += o3;
00103
                          np[iz]++;
00104
                       }
```

```
}
00106
          }
00107
00108
        /* Normalize... */
for (z = z0; z <= z1; z += dz) {
  iz = (int) ((z - z0) / dz);
00109
00110
00111
00112
          if (np[iz] > 0) {
00113
           timem[iz] /= np[iz];
            lonm[iz] /= np[iz];
latm[iz] /= np[iz];
00114
00115
            tm[iz] /= np[iz];
um[iz] /= np[iz];
00116
00117
00118
             vm[iz] /= np[iz];
00119
             wm[iz] /= np[iz];
00120
            h2om[iz] /= np[iz];
00121
             o3m[iz] /= np[iz];
         } else {
00122
            timem[iz] = GSL_NAN;
            lonm[iz] = GSL_NAN;
latm[iz] = GSL_NAN;
00124
00125
00126
             tm[iz] = GSL_NAN;
            um[iz] = GSL_NAN;
vm[iz] = GSL_NAN;
00127
00128
00129
             wm[iz] = GSL_NAN;
00130
            h2om[iz] = GSL_NAN;
00131
             o3m[iz] = GSL_NAN;
00132
00133
00134
00135
        /* Create output file... */
00136
        printf("Write meteorological data file: %s\n", argv[2]);
00137
        if (!(out = fopen(argv[2], "w")))
00138
          ERRMSG("Cannot create file!");
00139
        /* Write header... */
00140
00141
        fprintf(out,
                  "# $1 = time [s] \n"
00143
                 "# $2 = altitude [km] \n"
00144
                 "# $3 = longitude [deg] \n"
00145
                 "# $4 = latitude [deg]\n"
                 "# $5 = pressure [hPa]\n"
"# $6 = temperature [K]\n"
00146
00147
                 "# $7 = zonal wind [m/s]\n"
00148
                 "# $8 = meridional wind [m/s]\n"
00150
                 "# $9 = vertical wind [hPa/s] n"
                 "# $10 = H2O volume mixing ratio [1]\n"
"# $11 = O3 volume mixing ratio [1]\n\n");
00151
00152
00153
        /* Write data... */
00154
        for (z = z0; z \le z1; z += dz) {
00155
         iz = (int) ((z - z0) / dz);
00156
00157
          fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
00158
                    timem[iz], z, lonm[iz], latm[iz], P(z),
                    tm[iz], um[iz], vm[iz], wm[iz], h2om[iz], o3m[iz]);
00159
00160
00162
        /* Close file... */
00163
        fclose(out);
00164
00165
        /* Free... */
00166
        free (met);
00167
        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

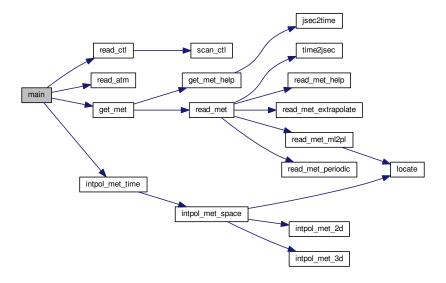
#### 5.21.2.1 int main ( int argc, char \* argv[] )

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

```
00033
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
        /* Allocate... */
00052
        ALLOC(atm, atm_t, 1);
00053
        ALLOC(met0, met_t, 1);
00054
        ALLOC(met1, met_t, 1);
00055
00056
        /* Read control parameters... */
00057
        read_ctl(argv[1], argc, argv, &ctl);
00058
00059
       /* 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
                       = time [s]\n"
00070
                "# $2
                      = altitude [km] \n"
00071
                "# $3
                      = longitude [deg]\n"
                "# $4 = latitude [deg]\n"
"# $5 = pressure [hPa]\n"
00072
00073
00074
                "# $6 = temperature [K]\n"
00075
                      = zonal wind [m/s]\n"
                "# $7
00076
                "# $8 = meridional wind [m/s]\n"
                "# $9 = vertical wind [hPa/s]\n"
"# $10 = H20 volume mixing ratio [1]\n"
00077
00078
                "# $11 = 03 volume mixing ratio [1]\n\n");
00079
08000
00081
        /* Loop over air parcels... */
00082
       for (ip = 0; ip < atm->np; ip++) {
00083
00084
          /* Get meteorological data... */
00085
         get_met(&ctl, argv[2], atm->time[ip], met0, met1);
00086
00087
          /* Interpolate meteorological data... */
00088
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
     lon[ip],
00089
                          atm->lat[ip], NULL, &t, &u, &v, &w, &h2o, &o3);
00090
         00091
00092
                 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
        /* Close file... */
00097
00098
       fclose(out);
00099
00100
00101
        free(atm);
00102
       free (met0);
00103
       free (met1);
00104
00105
        return EXIT_SUCCESS;
00106 }
```

5.22 met sample.c 137

Here is the call graph for this function:



# 5.22 met\_sample.c

```
00001 /*
00002
        This file is part of MPTRAC.
00003
00004
         MPTRAC is free software: you can redistribute it and/or modify
         it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
00005
00006
00007
         (at your option) any later version.
00008
00009
         MPTRAC is distributed in the hope that it will be useful,
00010
         but WITHOUT ANY WARRANTY; without even the implied warranty of
00011
         MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
         GNU General Public License for more details.
00013
         You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
00016
         Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00017
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /*
00028
          Main...
00029
00030
00031 int main(
00032
         int argc,
00033
        char *argv[]) {
00034
00035
         ctl_t ctl;
00036
00037
         atm t *atm;
00038
00039
         met_t *met0, *met1;
00040
00041
        FILE *out;
00042
00043
         double t, u, v, w, h2o, o3;
00044
00045
         int ip;
00046
00047
         /* Check arguments... */
00048
         if (argc < 4)
00049
           ERRMSG("Give parameters: <ctl> <metbase> <atm_in> <sample.tab>");
00050
00051
         /* Allocate... */
```

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

# 5.23 met\_zm.c File Reference

Extract zonal mean from meteorological data.

## **Functions**

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

## 5.23.1 Detailed Description

Extract zonal mean from meteorological data.

Definition in file met\_zm.c.

#### 5.23.2 Function Documentation

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

Definition at line 27 of file met zm.c.

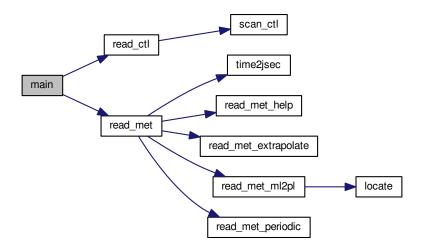
```
00029
00030
00031
                ctl_t ctl;
00032
00033
                met t *met;
00034
                FILE *in, *out;
00036
00037
                static double timem[EP][EY], psm[EP][EY], tm[EP][EY], um[EP][EY],
                   vm(EP][EY], vhm[EP][EY], wm[EP][EY], h2om[EP][EY], o3m[EP][EY],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

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

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

5.24 met zm.c 141

Here is the call graph for this function:



# 5.24 met\_zm.c

```
00001 /*
00002
         This file is part of MPTRAC.
00003
00004
         MPTRAC is free software: you can redistribute it and/or modify
00005
         it under the terms of the GNU General Public License as published by
00006
         the Free Software Foundation, either version 3 of the License, or
00007
         (at your option) any later version.
00008
00009
         MPTRAC is distributed in the hope that it will be useful,
00010
         but WITHOUT ANY WARRANTY; without even the implied warranty of
00011
         MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
         GNU General Public License for more details.
00013
00014
         You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00015
00016
00017
         Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028
         int argc,
00029
        char *argv[]) {
00030
00031
         ctl t ctl;
00032
00033
         met_t *met;
00034
00035
         FILE *in, *out;
00036
         static double timem[EP][EY], psm[EP][EY], tm[EP][EY], um[EP][EY],
00037
           vm[EP][EY], vhm[EP][EY], wm[EP][EY], h2om[EP][EY], o3m[EP][EY],
psm2[EP][EY], tm2[EP][EY], um2[EP][EY], vm2[EP][EY], vhm2[EP][EY],
wm2[EP][EY], h2om2[EP][EY], o3m2[EP][EY];
00038
00039
00040
00041
00042
         static int i, ip, ix, iy, np[EP][EY];
00043
00044
         /* Allocate... */
00045
         ALLOC(met, met_t, 1);
00046
00047
         /* Check arguments... */
00048
         if (argc < 4)
00049
           ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00050
00051
         /* Read control parameters... */
00052
         read_ctl(argv[1], argc, argv, &ctl);
```

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

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

# 5.25 smago.c File Reference

Estimate horizontal diffusivity based on Smagorinsky theory.

#### **Functions**

• int main (int argc, char \*argv[])

## 5.25.1 Detailed Description

Estimate horizontal diffusivity based on Smagorinsky theory.

Definition in file smago.c.

### 5.25.2 Function Documentation

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

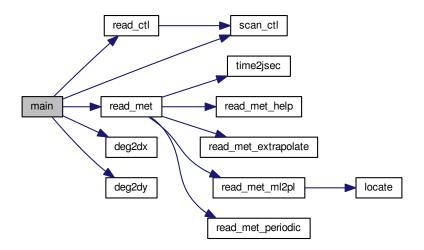
Definition at line 8 of file smago.c.

```
00010
00011
00012
        ctl_t ctl;
00013
00014
        met t *met;
00015
00016
00017
        static double dz, dzmin = 1e10, z, t, s, 1s2, k[EX][EY], c = 0.15;
00018
00019
00020
        static int ip, ip2, ix, iv;
00021
00022
         /* Allocate... */
00023
        ALLOC(met, met_t, 1);
00024
        /* Check arguments... */
if (argc < 4)</pre>
00025
00026
          ERRMSG("Give parameters: <ctl> <map.tab> <met>");
00027
00028
00029
        /* Read control parameters... */
        read_ctl(argv[1], argc, argv, &ctl);
z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00030
00031
00032
00033
        /* Read meteorological data... */
00034
        read_met(&ctl, argv[3], met);
```

```
00035
00036
         /* Find nearest pressure level... */
        for (ip2 = 0; ip2 < met->np; ip2++) {
  dz = fabs(Z(met->p[ip2]) - z);
  if (dz < dzmin) {</pre>
00037
00038
00039
            dzmin = dz;
00040
00041
            ip = ip2;
00042
00043
00044
        /* Write info... */
printf("Analyze %g hPa...\n", met->p[ip]);
00045
00046
00047
00048
         /* Calculate horizontal diffusion coefficients... */
        for (ix = 1; ix < met->nx - 1; ix++)
for (iy = 1; iy < met->ny - 1; iy++) {
00049
00050
             t = 0.5 * ((met->u[ix + 1][iy][ip] - met->u[ix - 1][iy][ip])
00051
                         / (1000. *
00052
00053
                            deg2dx (met->lon[ix + 1] - met->lon[ix - 1], met->
      lat[iy]))
00054
                         - (met->v[ix][iy + 1][ip] - met->v[ix][iy - 1][ip])
00055
                         / (1000. * deg2dy(met->lat[iy + 1] - met->lat[iy - 1])));
             00056
00057
00058
00059
                         / (1000. *
                            deg2dx(met->lon[ix + 1] - met->lon[ix - 1],
00060
00061
                                   met->lat[iy])));
            ls2 = gsl_pow_2(c * 500. * deg2dy(met->lat[iy + 1] - met->lat[iy - 1]));
if (fabs(met->lat[iy]) > 80)
00062
00063
               ls2 *= (90. - fabs(met->lat[iy])) / 10.;
00064
00065
             k[ix][iy] = 1s2 * sqrt(2.0 * (gsl_pow_2(t) + gsl_pow_2(s)));
00066
00067
        /* Create output file... */
printf("Write data file: %s\n", argv[2]);
if (!(out = fopen(argv[2], "w")))
00068
00069
00070
00071
           ERRMSG("Cannot create file!");
00072
00073
         /* Write header... */
        00074
00075
                 "# $2 = latitude [deg]\n"
00076
                 "# $3 = zonal wind [m/s]\n"
00077
00078
                 "# $4 = meridional wind [m/s]\n"
00079
                 "# $5 = horizontal diffusivity [m^2/s]\n");
08000
00081
        /* \ {\tt Write \ data...} \ */
        for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
  for (ix = 0; ix < met->nx; ix++)
00082
00083
00084
00085
             if (met \rightarrow lon[ix] >= 180)
               00086
00087
00088
00089
           for (ix = 0; ix < met->nx; ix++)
            if (met->lon[ix] <= 180)</pre>
00090
00091
               fprintf(out, "%g %g %g %g %g\n",
00092
                        met->lon[ix], met->lat[iy],
00093
                        met \rightarrow u[ix][iy][ip], met \rightarrow v[ix][iy][ip], k[ix][iy]);
00094
00095
00096
         /* Close file... */
00097
        fclose(out);
00098
         /* Free... */
00099
00100
        free (met);
00101
00102
        return EXIT_SUCCESS;
00103 }
```

5.26 smago.c 145

Here is the call graph for this function:



# 5.26 smago.c

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

```
00052
                         / (1000. *
                             deg2dx(met->lon[ix + 1] - met->lon[ix - 1], met->
      lat[iy]))
00054
                         - (met->v[ix][iy + 1][ip] - met->v[ix][iy - 1][ip])
             / (1000. * deg2dy(met->lat[iy + 1] - met->lat[iy - 1])));

s = 0.5 * ((met->u[ix][iy + 1][ip] - met->u[ix][iy - 1][ip])

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

# 5.27 split.c File Reference

Split air parcels into a larger number of parcels.

### **Functions**

• int main (int argc, char \*argv[])

### 5.27.1 Detailed Description

Split air parcels into a larger number of parcels.

Definition in file split.c.

### 5.27.2 Function Documentation

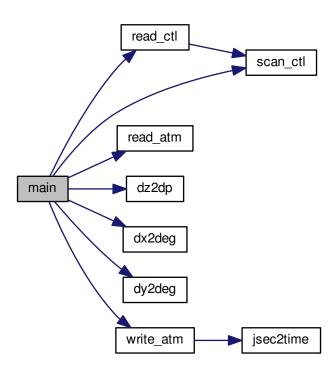
### 5.27.2.1 int main ( int argc, char \* argv[])

Definition at line 27 of file split.c.

```
00029
00030
00031
          atm_t *atm, *atm2;
00032
00033
          ctl t ctl;
00034
          gsl_rng *rng;
00036
00037
          double m, mtot = 0, dt, dx, dz, mmax = 0,
00038
            t0, t1, z0, z1, lon0, lon1, lat0, lat1;
00039
00040
          int i, ip, iq, n;
00041
00042
          /* Allocate... */
00043
          ALLOC(atm, atm_t, 1);
00044
          ALLOC(atm2, atm_t, 1);
00045
00046
          /* Check arguments... */
00047
              (argc < 4)
00048
             ERRMSG("Give parameters: <ctl> <atm_in> <atm_out>");
00049
00050
          /* Read control parameters... */
         read_ctl(argv[1], argc, argv, &ctl);
n = (int) scan_ctl(argv[1], argc, argv, "SPLIT_N", -1, "", NULL);
m = scan_ctl(argv[1], argc, argv, "SPLIT_M", -1, "-999", NULL);
dt = scan_ctl(argv[1], argc, argv, "SPLIT_DT", -1, "0", NULL);
00051
00052
00053
          to = scan_ctl(argv[1], argc, argv, "SPLIT_TO", -1, "0", NULL);
t1 = scan_ctl(argv[1], argc, argv, "SPLIT_T1", -1, "0", NULL);
00055
00056
          dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);
z0 = scan_ctl(argv[1], argc, argv, "SPLIT_ZO", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "SPLIT_Z1", -1, "0", NULL);
00057
00058
00059
          dx = scan_ctl(argv[1], argc, argv, "SPLIT_DX", -1, "0", NULL);
         lon0 = scan_ctl(argv[1], argc, argv, "SPLIT_LON0", -1, "0", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "SPLIT_LON1", -1, "0", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT0", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT1", -1, "0", NULL);
00061
00062
00063
00064
00065
00066
          /* Init random number generator... */
          gsl_rng_env_setup();
00067
00068
          rng = gsl_rng_alloc(gsl_rng_default);
00069
00070
         /* Read atmospheric data... */
00071
          read_atm(argv[2], &ctl, atm);
00072
00073
          /\star Get total and maximum mass... \star/
00074
          if (ctl.qnt_m >= 0)
00075
            for (ip = 0; ip < atm->np; ip++) {
               mtot += atm->q[ctl.qnt_m][ip];
00076
00077
               mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
00078
00079
          if (m > 0)
00080
            mtot = m;
00081
00082
          /* Loop over air parcels... */
00083
          for (i = 0; i < n; i++) {
00084
00085
             /* Select air parcel... */
00086
             if (ctl.qnt_m >= 0)
00087
               do {
00088
                 ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00089
               } while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
00090
             else
00091
               ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00092
00093
             /* Set time... */
00094
             if (t1 > t0)
00095
               atm2 - time[atm2 - np] = t0 + (t1 - t0) * qsl_rnq_uniform_pos(rnq);
00096
             else
00097
              atm2->time[atm2->np] = atm->time[ip]
00098
                 + gsl_ran_gaussian_ziggurat(rng, dt / 2.3548);
00099
00100
             /* Set vertical position... */
             if (z1 > z0)
00101
00102
               atm2 \rightarrow p[atm2 \rightarrow np] = P(z0 + (z1 - z0) * gsl_rng_uniform_pos(rng));
00103
00104
               atm2->p[atm2->np] = atm->p[ip]
```

```
+ dz2dp(gsl_ran_gaussian_ziggurat(rng, dz / 2.3548), atm->p[ip]);
00106
            /* Set horizontal position... */
if (lon1 > lon0 && lat1 > lat0) {
00107
00108
             atm2->lon(atm2->np) = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
atm2->lat(atm2->np) = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
00109
00110
00111
00112
             atm2->lon[atm2->np] = atm->lon[ip]
00113
                 + gsl_ran_gaussian_ziggurat(rng, dx2deg(dx, atm->lat[ip]) / 2.3548);
00114
              atm2->lat[atm2->np] = atm->lat[ip]
                 + gsl_ran_gaussian_ziggurat(rng, dy2deg(dx) / 2.3548);
00115
00116
00117
00118
            /* Copy quantities... */
            for (iq = 0; iq < ctl.nq; iq++)
  atm2->q[iq][atm2->np] = atm->q[iq][ip];
00119
00120
00121
           /* Adjust mass... */
if (ctl.qnt_m >= 0)
00122
00123
00124
              atm2->q[ctl.qnt_m][atm2->np] = mtot / n;
00125
           /* Increment particle counter... */
if ((++atm2->np) >= NP)
00126
00127
              ERRMSG("Too many air parcels!");
00128
00129
00130
00131
          /\star Save data and close file... \star/
00132
         write_atm(argv[3], &ctl, atm2, atm->time[0]);
00133
         /* Free... */
00134
00135
         free(atm);
00136
         free(atm2);
00137
00138
         return EXIT_SUCCESS;
00139 }
```

Here is the call graph for this function:



5.28 split.c 149

## 5.28 split.c

```
00001 /*
          This file is part of MPTRAC.
00003
00004
          MPTRAC is free software: you can redistribute it and/or modify
00005
          it under the terms of the GNU General Public License as published by
          the Free Software Foundation, either version 3 of the License, or
00006
00007
          (at your option) any later version.
80000
00009
          MPTRAC is distributed in the hope that it will be useful,
00010
          but WITHOUT ANY WARRANTY; without even the implied warranty of
00011
          MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
          GNU General Public License for more details.
00013
          You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
00016
00017
          Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028
         int argc,
00029
          char *argv[]) {
00030
00031
          atm t *atm, *atm2;
00032
00033
          ctl_t ctl;
00034
00035
          gsl_rng *rng;
00036
          double m, mtot = 0, dt, dx, dz, mmax = 0,
   t0, t1, z0, z1, lon0, lon1, lat0, lat1;
00037
00038
00039
00040
          int i, ip, iq, n;
00041
           /* Allocate... */
00042
          ALLOC(atm, atm_t, 1);
ALLOC(atm2, atm_t, 1);
00043
00044
00045
00046
          /* Check arguments... */
00047
          if (argc < 4)
00048
             ERRMSG("Give parameters: <ctl> <atm_in> <atm_out>");
00049
00050
          /* Read control parameters... */
          /* Read control parameters... */
read_ctl(argv[1], argc, argv, &ctl);
n = (int) scan_ctl(argv[1], argc, argv, "SPLIT_N", -1, "", NULL);
m = scan_ctl(argv[1], argc, argv, "SPLIT_M", -1, "-999", NULL);
dt = scan_ctl(argv[1], argc, argv, "SPLIT_DT", -1, "0", NULL);
t0 = scan_ctl(argv[1], argc, argv, "SPLIT_TO", -1, "0", NULL);
t1 = scan_ctl(argv[1], argc, argv, "SPLIT_TI", -1, "0", NULL);
dr = scan_ctl(argv[1], argc, argv, "SPLIT_DT", -1, "0", NULL);
00051
00052
00053
00054
00055
00056
00057
          dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);
          z0 = scan_ctl(argv[1], argc, argv, "SPLIT_Z0", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "SPLIT_Z1", -1, "0", NULL);
00058
00059
           dx = scan_ctl(argv[1], argc, argv, "SPLIT_DX", -1, "0", NULL);
00060
          lon0 = scan_ctl(argv[1], argc, argv, "SPLIT_LON0", -1, "0", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "SPLIT_LON1", -1, "0", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT0", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT1", -1, "0", NULL);
00061
00062
00063
00064
00065
00066
           /* Init random number generator... */
00067
          gsl_rng_env_setup();
00068
          rng = gsl_rng_alloc(gsl_rng_default);
00069
00070
           /* Read atmospheric data... */
00071
          read_atm(argv[2], &ctl, atm);
00072
00073
           /* Get total and maximum mass... */
00074
          if (ctl.qnt_m >= 0)
            for (ip = 0; ip < atm->np; ip++) {
00075
               mtot += atm->q[ctl.qnt_m][ip];
00076
00077
                mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
00078
          if (m > 0)
00079
00080
            mtot = m:
00081
00082
           /* Loop over air parcels... */
          for (i = 0; i < n; i++) {
00083
00084
00085
              /* Select air parcel... */
             if (ctl.qnt_m >= 0)
00086
00087
               do {
00088
                  ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
                } while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
```

```
00090
00091
             ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00092
            /* Set time... */
00093
00094
           if (t1 > t0)
00095
             atm2 \rightarrow time[atm2 \rightarrow np] = t0 + (t1 - t0) * qsl_rnq_uniform_pos(rnq);
00096
            else
00097
             atm2->time[atm2->np] = atm->time[ip]
00098
                + gsl_ran_gaussian_ziggurat(rng, dt / 2.3548);
00099
00100
           /* Set vertical position... */
00101
           if (z1 > z0)
00102
             atm2 - p[atm2 - np] = P(z0 + (z1 - z0) * gsl_rng_uniform_pos(rng));
00103
00104
             atm2->p[atm2->np] = atm->p[ip]
00105
                + dz2dp(gsl\_ran\_gaussian\_ziggurat(rng, dz / 2.3548), atm->p[ip]);
00106
           /* Set horizontal position... */
if (lon1 > lon0 && lat1 > lat0) {
00107
00108
             atm2->lon[atm2->np] = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
atm2->lat[atm2->np] = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
00109
00110
00111
00112
             atm2 - lon[atm2 - lon[ip]] = atm - lon[ip]
             + gsl_ran_gaussian_ziggurat(rng, dx2deg(dx, atm->lat[ip]) / 2.3548);
atm2->lat[atm2->np] = atm->lat[ip]
00113
00114
00115
               + gsl_ran_gaussian_ziggurat(rng, dy2deg(dx) / 2.3548);
00116
00117
00118
           /* Copy quantities... */
           for (iq = 0; iq < ctl.nq; iq++)
  atm2->q[iq][atm2->np] = atm->q[iq][ip];
00119
00120
00121
00122
            /* Adjust mass...
00123
           if (ctl.qnt_m >= 0)
00124
             atm2->q[ctl.qnt_m][atm2->np] = mtot / n;
00125
          /* Increment particle counter... */
if ((++atm2->np) >= NP)
00126
00128
              ERRMSG("Too many air parcels!");
00129
00130
         /* Save data and close file... */
00131
00132
         write_atm(argv[3], &ctl, atm2, atm->time[0]);
00133
00134
         /* Free... */
00135
         free(atm);
00136
        free(atm2);
00137
         return EXIT_SUCCESS;
00138
00139 }
```

# 5.29 time2jsec.c File Reference

Convert date to Julian seconds.

### **Functions**

• int main (int argc, char \*argv[])

# 5.29.1 Detailed Description

Convert date to Julian seconds.

Definition in file time2jsec.c.

5.30 time2jsec.c 151

### 5.29.2 Function Documentation

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

Definition at line 27 of file time2jsec.c.

```
00029
00030
00031
        double jsec, remain;
00032
00033
        int day, hour, min, mon, sec, year;
00034
00035
         /* Check arguments... */
00036
          ERRMSG("Give parameters: <year> <mon> <day> <hour> <min> <sec> <remain>");
00037
00038
00039
        /* Read arguments... */
00040
        year = atoi(argv[1]);
00041
        mon = atoi(argv[2]);
00042
        day = atoi(argv[3]);
00043
        hour = atoi(argv[4]);
00044
        min = atoi(argv[5]);
sec = atoi(argv[6]);
00045
00046
        remain = atof(argv[7]);
00047
00048
        time2jsec(year, mon, day, hour, min, sec, remain, &jsec);
printf("%.2f\n", jsec);
00049
00050
00051
00052
        return EXIT_SUCCESS;
00053 }
```

Here is the call graph for this function:



# 5.30 time2jsec.c

```
00001 /*
00002
        This file is part of MPTRAC.
00003
        MPTRAC is free software: you can redistribute it and/or modify
00004
         it under the terms of the GNU General Public License as published by
00005
00006
        the Free Software Foundation, either version 3 of the License, or
00007
         (at your option) any later version.
80000
00009
        MPTRAC is distributed in the hope that it will be useful,
        but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
00012
        GNU General Public License for more details.
00013
00014
        You should have received a copy of the GNU General Public License
00015
        along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/>.
00016
00017
        Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028
      int argc,
00029
        char *argv[]) {
```

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

## 5.31 trac.c File Reference

Lagrangian particle dispersion model.

### **Functions**

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

Set simulation time interval.

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

Calculate advection of air parcels.

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

Calculate exponential decay of particle mass.

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

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

Calculate turbulent diffusion.

void module\_isosurf (ctl\_t \*ctl, met\_t \*met0, met\_t \*met1, atm\_t \*atm, int ip)

Force air parcels to stay on isosurface.

• void module\_meteo (ctl\_t \*ctl, met\_t \*met0, met\_t \*met1, atm\_t \*atm, int ip)

Interpolate meteorological data for air parcel positions.

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

Check position of air parcels.

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

Calculate sedimentation of air parcels.

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

Write simulation output.

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

# 5.31.1 Detailed Description

Lagrangian particle dispersion model.

Definition in file trac.c.

5.31 trac.c File Reference 153

### 5.31.2 Function Documentation

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

Set simulation time interval.

Definition at line 398 of file trac.c.

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

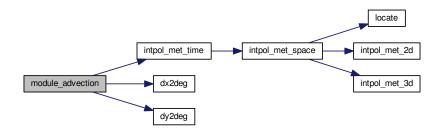
5.31.2.2 void module\_advection ( met\_t \* met0, met\_t \* met1, atm\_t \* atm, int ip, double dt )

Calculate advection of air parcels.

Definition at line 422 of file trac.c.

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

Here is the call graph for this function:



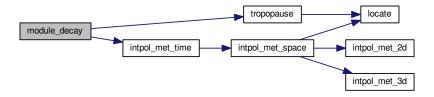
5.31.2.3 void module\_decay (  $ctl_t * ctl$ ,  $met_t * met0$ ,  $met_t * met1$ ,  $atm_t * atm$ , int ip, double dt)

Calculate exponential decay of particle mass.

Definition at line 455 of file trac.c.

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

Here is the call graph for this function:



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

Calculate mesoscale diffusion.

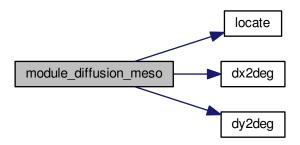
Definition at line 497 of file trac.c.

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

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

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

Here is the call graph for this function:



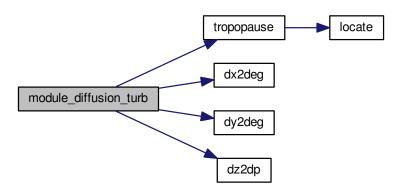
 $5.31.2.5 \quad \text{void module\_diffusion\_turb ( } \textbf{ctl\_t} * \textit{ctl, } \textbf{atm\_t} * \textit{atm, } \textbf{int } \textit{ip, } \textbf{double } \textit{dt, } \textbf{gsl\_rng} * \textit{rng } \textbf{)}$ 

Calculate turbulent diffusion.

Definition at line 608 of file trac.c.

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

Here is the call graph for this function:



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

Force air parcels to stay on isosurface.

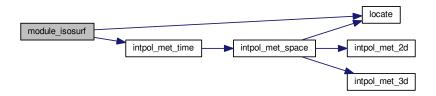
Definition at line 653 of file trac.c.

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

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

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

Here is the call graph for this function:



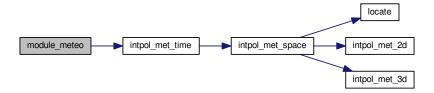
```
5.31.2.7 void module_meteo ( ctl t * ctl, met t * met0, met t * met1, atm t * atm, int ip )
```

Interpolate meteorological data for air parcel positions.

Definition at line 768 of file trac.c.

```
00773
                 {
00774
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:



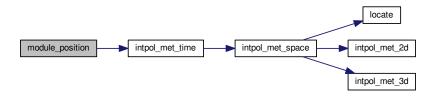
```
5.31.2.8 void module_position ( met_t * met0, met_t * met1, atm_t * atm, int ip )
```

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;
00852
00853
00854
          while (atm->lon[ip] >= 180)
atm->lon[ip] -= 360;
00855
00856
00857
00858
          /* Get surface pressure... */
          00859
00860
00861
                                NULL, NULL, NULL, NULL, NULL);
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:



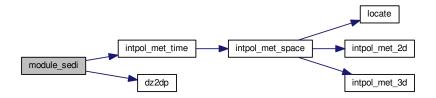
5.31.2.9 void module\_sedi (  $ctl_t * ctl$ ,  $met_t * met0$ ,  $met_t * met1$ ,  $atm_t * atm$ , int ip, double dt)

Calculate sedimentation of air parcels.

Definition at line 872 of file trac.c.

```
00878
00879
         /* 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
        /\star Check if parameters are available... \star/
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];
00898
        rho_p = atm->q[ctl->qnt_rho][ip];
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
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:



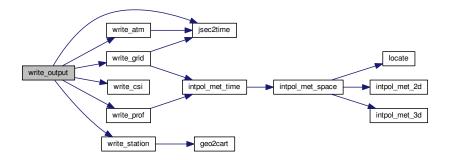
5.31.2.10 void write\_output ( const char \* dirname, ctl t \* ctl, met t \* met0, met t \* met1, atm t \* atm, double t )

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_%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...
          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... */
          /* 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
          /* Write station data... */
if (ctl->stat_basename[0] != '-') {
   sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
00976
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[])

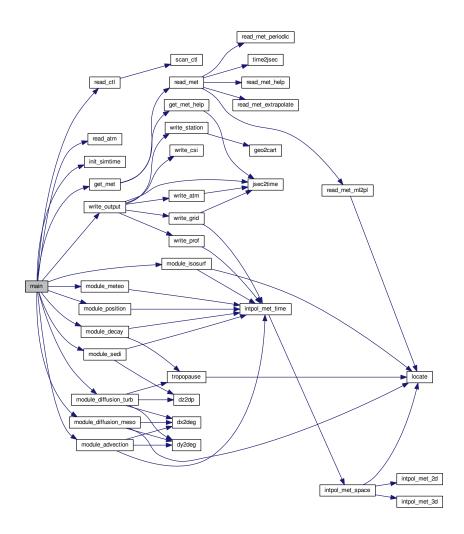
Definition at line 160 of file trac.c.

```
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
        /* \ {\tt 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
          /* Read control parameters... */
00218
          \label{eq:sprintf} \mbox{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
             /* 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... */
00264
             START_TIMER(TIMER_INPUT);
00265
             get_met(&ctl, argv[4], t, met0, met1);
00266
             STOP_TIMER(TIMER_INPUT);
00267
             /* Initialize isosurface... */
00268
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
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)
            for (ip = 0; ip < atm->np; ip++)
00294
00295
              if (gsl_finite(dt[ip]))
00296
                 module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
00297
                                         rng[omp_get_thread_num()]);
00298
            STOP_TIMER(TIMER_DIFFMESO);
00299
             /* 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
              /* Meteorological data... */
00322
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)
             for (ip = 0; ip < atm->np; ip++)
00332
              if (gsl_finite(dt[ip]))
00333
00334
                  module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00335
             STOP_TIMER(TIMER_DECAY);
00336
             /* Write output... */
00337
             START_TIMER (TIMER_OUTPUT);
00338
             write_output(dirname, &ctl, met0, met1, atm, t);
00340
             STOP_TIMER(TIMER_OUTPUT);
00341
00342
00343
00344
             Finalize model run...
00345
00346
00347
00348
           STOP_TIMER(TIMER_TOTAL);
00349
           PRINT_TIMER(TIMER_TOTAL);
           PRINT_TIMER(TIMER_INIT);
PRINT_TIMER(TIMER_INPUT);
00350
00351
           PRINT_TIMER (TIMER_OUTPUT);
00352
00353
           PRINT_TIMER (TIMER_ADVECT);
00354
           PRINT_TIMER (TIMER_DECAY);
00355
           PRINT_TIMER (TIMER_DIFFMESO);
00356
           PRINT_TIMER (TIMER_DIFFTURB);
00357
           PRINT_TIMER (TIMER_ISOSURF);
           PRINT_TIMER (TIMER_METEO);
00359
           PRINT_TIMER (TIMER_POSITION);
00360
           PRINT_TIMER (TIMER_SEDI);
00361
00362
           /* Report memory usage... */
           00363
00364
00365
00366
            \begin{aligned} & \text{printf("MEMORY\_STATIC = $g MByte\n",} \\ & \text{(((EX + EY) + (2 + NQ) * GX * GY * GZ) * sizeof(double)} \\ & \text{+ (EX * EY + EX * EY * EP) * sizeof(float)} \end{aligned} 
00367
00368
00369
                    + (2 * GX * GY * GZ) * sizeof(int)) / 1024. / 1024.);
00370
00371
           /* Report problem size... */
printf("SIZE_NP = %d\n", atm->np);
00372
00373
           printf("SIZE_TASKS = %d\n", size);
printf("SIZE_THREADS = %d\n", omp_get_max_threads());
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
```

Here is the call graph for this function:



## 5.32 trac.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
            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
```

5.32 trac.c 167

```
00018 */
00019
00025 #include "libtrac.h"
00026
00027 #ifdef MPT
00028 #include "mpi.h"
00029 #endif
00030
00031 /* -----
         Defines...
00032
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
00049
00051 #define TIMER_DECAY 5
00052
00054 #define TIMER_DIFFMESO 6
00055
00057 #define TIMER_DIFFTURB 7
00058
00060 #define TIMER_ISOSURF 8
00061
00063 #define TIMER_METEO 9
00064
00066 #define TIMER_POSITION 10
00067
00069 #define TIMER_SEDI 11
00070
00071 /* -
       Functions...
00073
00074
00076 void init_simtime(
00077 ctl_t * ctl,
00078 atm_t * atm);
00079
00081 void module_advection(
00082
        met_t * met0,
        met_t * met1,
atm_t * atm,
00083
00084
00085
        int ip,
00086
        double dt);
00087
00089 void module_decay(
00090
        ctl_t * ctl,
         met_t * met0,
00091
        met_t * met1,
atm_t * atm,
00092
00093
00094
         int ip,
00095
        double dt);
00096
00098 void module_diffusion_meso(
        ctl_t * ctl,
met_t * met0,
met_t * met1,
00099
00100
00101
00102
        atm_t * atm,
00103
         int ip,
00104
        double dt,
00105
        gsl_rng * rng);
00106
00108 void module_diffusion_turb(
        ctl_t * ctl,
atm_t * atm,
00109
00110
00111
         int ip,
00112
        double dt,
00113
        gsl_rng * rng);
00114
00116 void module_isosurf(
00117
        ctl_t * ctl,
00118
         met_t * met0,
        met_t * met1,
atm_t * atm,
00119
00120
00121
         int ip);
00122
00124 void module_meteo(
00125
        ctl_t * ctl,
        met_t * met0,
met_t * met1,
atm_t * atm,
00126
00127
00128
```

```
00129
        int ip);
00130
00132 void module_position(
00133
        met_t * met0,
        met_t * met1,
atm_t * atm,
00134
00135
00136
        int ip);
00137
00139 void module_sedi(
        ctl_t * ctl,
met_t * met0,
00140
00141
00142
        met_t * met1,
        atm_t * atm,
00143
00144
        int ip,
00145
        double dt);
00146
00148 void write_output(
00149
        const char *dirname,
        ctl_t * ctl,
met_t * met0,
00150
00151
00152
        met_t * met1,
00153
        atm_t * atm,
        double t);
00154
00155
00156 /*
00157
       Main...
00158
00159
00160 int main(
00161
        int argc,
00162
        char *argv[]) {
00163
00164
        ctl_t ctl;
00165
00166
        atm_t *atm;
00167
00168
        met t *met0, *met1;
00169
00170
        gsl_rng *rng[NTHREADS];
00171
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
        /* Open directory list... */
if (!(dirlist = fopen(argv[1], "r")))
00191
00192
          ERRMSG("Cannot open directory list!");
00193
00194
        /* Loop over directories... */
while (fscanf(dirlist, "%s", dirname) != EOF) {
00195
00196
00197
           /\star MPI parallelization... \star/
00198
00199
           if ((++ntask) % size != rank)
00200
             continue;
00201
00202
00203
             Initialize model run...
00204
00205
00206
           /* Set timers... */
00207
           START_TIMER(TIMER_TOTAL);
00208
           START_TIMER (TIMER_INIT);
00209
00210
           /* Allocate... */
           ALLOC(atm, atm_t, 1);
ALLOC(met0, met_t, 1);
ALLOC(met1, met_t, 1);
00211
00212
00213
00214
           ALLOC(dt, double,
00215
                 NP);
00216
           /* Read control parameters... */
sprintf(filename, "%s/%s", dirname, argv[2]);
00217
00218
```

5.32 trac.c 169

```
read_ctl(filename, argc, argv, &ctl);
00220
00221
          /* Initialize random number generators... */
00222
          gsl_rng_env_setup();
          for (i = 0; i < NTHREADS; i++)</pre>
00223
00224
            rng[i] = gsl_rng_alloc(gsl_rng_default);
00226
           /* Read atmospheric data... */
00227
          sprintf(filename, "%s/%s", dirname, argv[3]);
00228
          read_atm(filename, &ctl, atm);
00229
00230
           /* Get simulation time interval... */
00231
          init_simtime(&ctl, atm);
00232
00233
           /\star Get rounded start time... \star/
          if (ctl.direction == 1)
  t0 = floor(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00234
00235
00236
          else
00237
            t0 = ceil(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00238
00239
           /* Set timers... */
00240
          STOP_TIMER(TIMER_INIT);
00241
00242
00243
             Loop over timesteps...
00244
00245
           /* Loop over timesteps... */
00246
          for (t = t0; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;</pre>
00247
                t += ctl.direction * ctl.dt_mod) {
00248
00249
00250
             /* Adjust length of final time step... */
00251
            if (ctl.direction * (t - ctl.t_stop) > 0)
00252
               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
                    && 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
                dt[ip] = GSL_NAN;
00261
00262
00263
             /* Get meteorological data... */
00264
             START_TIMER(TIMER_INPUT);
00265
             get_met(&ctl, argv[4], t, met0, met1);
00266
             STOP_TIMER(TIMER_INPUT);
00267
             /* Initialize isosurface... */
00268
00269
             START_TIMER(TIMER_ISOSURF);
00270
            if (t == t0)
00271
               module_isosurf(&ctl, met0, met1, atm, -1);
00272
             STOP_TIMER(TIMER_ISOSURF);
00273
00274
             /* Advection...
            START_TIMER(TIMER_ADVECT);
00276 #pragma omp parallel for default(shared) private(ip)
            for (ip = 0; ip < atm->np; ip++)
00277
00278
              if (gsl_finite(dt[ip]))
                module_advection(met0, met1, atm, ip, dt[ip]);
00279
00280
            STOP TIMER (TIMER ADVECT);
00281
00282
             /* Turbulent diffusion..
00283
             START_TIMER(TIMER_DIFFTURB);
00284 #pragma omp parallel for default(shared) private(ip)
00285
            for (ip = 0; ip < atm->np; ip++)
              if (gsl_finite(dt[ip]))
00286
                module_diffusion_turb(&ctl, atm, ip, dt[ip],
00287
                                         rng[omp_get_thread_num()]);
00288
00289
             STOP_TIMER(TIMER_DIFFTURB);
00290
            /* Mesoscale diffusion... */
START_TIMER(TIMER_DIFFMESO);
00291
00292
00293 #pragma omp parallel for default(shared) private(ip)
            for (ip = 0; ip < atm->np; ip++)
00294
00295
              if (gsl_finite(dt[ip]))
00296
                module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
00297
                                        rng[omp_get_thread_num()]);
            STOP TIMER (TIMER DIFFMESO):
00298
00299
00300
             /* Sedimentation...
             START_TIMER(TIMER_SEDI);
00301
00302 #pragma omp parallel for default(shared) private(ip)
00303
            for (ip = 0; ip < atm->np; ip++)
00304
              if (gsl_finite(dt[ip]))
00305
                 module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
```

```
00306
           STOP_TIMER(TIMER_SEDI);
00307
00308
            /* Isosurface... */
00309
           START_TIMER(TIMER_ISOSURF);
00310 \#pragma omp parallel for default(shared) private(ip)
           for (ip = 0; ip < atm->np; ip++)
00311
              module_isosurf(&ctl, met0, met1, atm, ip);
00313
            STOP_TIMER(TIMER_ISOSURF);
00314
00315
            /* Position...
           START_TIMER(TIMER_POSITION);
00316
00317 #pragma omp parallel for default(shared) private(ip)
           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++)
00325
00326
              module_meteo(&ctl, met0, met1, atm, ip);
00327
            STOP_TIMER(TIMER_METEO);
00328
00329
            /* Decay...
00330
            START_TIMER(TIMER_DECAY);
00331 #pragma omp parallel for default(shared) private(ip)
00332
            for (ip = 0; ip < atm->np; ip++)
00333
             if (gsl_finite(dt[ip]))
00334
               module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
            STOP_TIMER(TIMER_DECAY);
00335
00336
00337
            /* Write output...
00338
            START_TIMER(TIMER_OUTPUT);
00339
            write_output(dirname, &ctl, met0, met1, atm, t);
00340
            STOP_TIMER(TIMER_OUTPUT);
00341
00342
00344
            Finalize model run...
00345
00346
00347
          /* Report timers... */
          STOP_TIMER(TIMER_TOTAL);
00348
00349
          PRINT_TIMER (TIMER_TOTAL);
00350
          PRINT_TIMER(TIMER_INIT);
00351
          PRINT_TIMER (TIMER_INPUT);
00352
          PRINT_TIMER(TIMER_OUTPUT);
00353
          PRINT_TIMER (TIMER_ADVECT);
          PRINT_TIMER(TIMER_DECAY);
PRINT_TIMER(TIMER_DIFFMESO);
00354
00355
          PRINT_TIMER(TIMER_DIFFTURB);
00356
00357
          PRINT_TIMER(TIMER_ISOSURF);
00358
          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
00370
                  + (2 * GX * GY * GZ) * sizeof(int)) / 1024. / 1024.);
00371
         /* Report problem size... */
printf("SIZE_NP = %d\n", atm->np);
00372
00373
          printf("SIZE_TASKS = %d\n", size);
00374
00375
          printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00376
00377
          /\star Free random number generators... \star/
00378
          for (i = 0; i < NTHREADS; i++)
           gsl_rng_free(rng[i]);
00379
00380
00381
          /* Free... */
00382
          free(atm);
00383
          free (met0);
00384
          free (met1);
         free(dt);
00385
00386
00387
00388 #ifdef MPI
00389
     /* Finalize MPI... */
00390
       MPI_Finalize();
00391 #endif
00392
```

5.32 trac.c 171

```
return EXIT_SUCCESS;
00394 }
00395
00397
00398 void init_simtime(
00399
     ctl_t * ctl,
00400
       atm_t * atm)
00401
00402
       /\star Set inital and final time... \star/
       if (ctl->direction == 1) {
   if (ctl->t_start < -1e99)</pre>
00403
00404
00405
           ctl->t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00406
          if (ctl->t_stop < -1e99)
00407
           ctl->t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00408
         else if (ctl->direction == -1) {
00409
         if (ctl->t_stop < -1e99)</pre>
          ctl->t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
if (ctl->t_start < -1e99)</pre>
00410
00411
00412
           ctl->t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00413
00414
00415
        /* Check time... */
       if (ctl->direction * (ctl->t_stop - ctl->t_start) <= 0)</pre>
00416
00417
         ERRMSG("Nothing to do!");
00418 }
00419
00421
00422 void module advection(
00423
       met_t * met0,
00424
       met_t * met1,
00425
       atm_t * atm,
       int ip,
00426
00427
       double dt) {
00428
00429
       double v[3], xm[3];
00430
00431
        /* Interpolate meteorological data... */
00432
       intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00433
                        atm->lon[ip], atm->lat[ip], NULL, NULL,
00434
                        &v[0], &v[1], &v[2], NULL, NULL);
00435
00436
        /\star Get position of the mid point... \star/
       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... */
       intpol_met_time(met0, met1, atm->time[ip] + 0.5 * dt,
00442
                       xm[2], xm[0], xm[1], NULL, NULL,
00443
00444
                        &v[0], &v[1], &v[2], NULL, NULL);
00445
00446
       /* Save new position... */
       atm->time[ip] += dt;
atm->lon[ip] += dx2deg(dt * v[0] / 1000., xm[1]);
atm->lat[ip] += dy2deg(dt * v[1] / 1000.);
00447
00448
00450
       atm->p[ip] += dt * v[2];
00451 }
00452
00454
00455 void module_decay(
00456
       ctl_t * ctl,
00457
       met_t * met0,
00458
       met_t * met1,
       atm_t * atm,
00459
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)
         tdec = ctl->tdec_trop;
00471
00472
00473
       /* Set altitude-dependent lifetime... */
00474
       else {
00475
          /* Get surface pressure... */
00476
          intpol\_met\_time(met0, met1, atm->time[ip], atm->p[ip],
00477
00478
                         atm->lon[ip], atm->lat[ip], &ps, NULL,
```

```
NULL, NULL, NULL, NULL, NULL);
00479
00480
00481
           /* Get tropopause pressure... */
00482
           pt = tropopause(atm->time[ip], atm->lat[ip]);
00483
00484
           /* Set lifetime... */
           if (atm->p[ip] <= pt)</pre>
00486
             tdec = ctl->tdec_strat;
00487
00488
             tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
      p[ip]);
00489
00490
00491
         /* Calculate exponential decay... */
00492
         atm \rightarrow q[ctl \rightarrow qnt_m][ip] *= exp(-dt / tdec);
00493 }
00494
00495 /
        ******************************
00496
00497 void module_diffusion_meso(
        ctl_t * ctl,
met_t * met0,
00498
00499
00500
         met_t * met1,
         atm_t * atm,
00501
00502
         int ip,
00503
         double dt,
         gsl_rng * rng) {
00504
00505
00506
        double r, rs, u[16], v[16], w[16], usig, vsig, wsig;
00507
00508
        int ix, iv, iz:
00509
00510
         /* Calculate mesoscale velocity fluctuations... */
00511
         if (ctl->turb_meso > 0) {
00512
00513
           /* Get indices... */
           ix = locate(met0->lon, met0->nx, atm->lon[ip]);
00514
           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
00521
           u[2] = met0 -> u[ix][iy + 1][iz];
           u[3] = met0->u[ix + 1][iy + 1][iz];
u[4] = met0->u[ix][iy][iz + 1];
00522
00523
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];
00529
           v[1] = met0 -> v[ix + 1][iy][iz];
00530
           v[2] = met0 -> v[ix][iy + 1][iz];
           v[3] = met0->v[ix + 1][iy + 1][iz];
v[4] = met0->v[ix][iy][iz + 1];
00531
00532
           v[5] = met0->v[ix + 1][iy][iz + 1];
v[6] = met0->v[ix][iy + 1][iz + 1];
00533
00535
           v[7] = met0 -> v[ix + 1][iy + 1][iz + 1];
00536
00537
           w[0] = met0->w[ix][iy][iz];
           w[1] = met0->w[ix + 1][iy][iz];
00538
           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];
           w[7] = met0 -> w[ix + 1][iy + 1][iz + 1];
00544
00545
00546
           /* Get indices... */
           ix = locate(met1->lon, met1->nx, atm->lon[ip]);
00547
00548
           iy = locate(met1->lat, met1->ny, atm->lat[ip]);
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
00554
           u[10] = met1->u[ix][iy + 1][iz];
           u[11] = met1 - u[ix + 1][iy + 1][iz];

u[12] = met1 - u[ix][iy][iz + 1];
00555
00556
           u[13] = met1->u[ix + 1][iy][iz + 1];
u[14] = met1->u[ix][iy + 1][iz + 1];
00557
00558
           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];
           v[10] = met1->v[ix][iy + 1][iz];
v[11] = met1->v[ix + 1][iy + 1][iz];
00563
00564
```

5.32 trac.c 173

```
v[12] = met1->v[ix][iy][iz + 1];
          v[13] = met1->v[ix + 1][iy][iz + 1];
v[14] = met1->v[ix][iy + 1][iz + 1];
00566
00567
00568
          v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00569
          w[8] = met1->w[ix][iy][iz];
w[9] = met1->w[ix + 1][iy][iz];
00570
00571
00572
          w[10] = met1->w[ix][iy +
00573
          w[11] = met1->w[ix + 1][iy + 1][iz];
          w[12] = met1->w[ix][iy][iz + 1];
00574
          w[13] = met1->w[ix + 1][iy][iz + 1];
w[14] = met1->w[ix][iy + 1][iz + 1];
00575
00576
          w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00577
00578
00579
           /* Get standard deviations of local wind data... */
          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
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,
00591
                                                                  ctl->turb_meso * usig);
00592
          atm->vp[ip] =
00593
            r * atm->vp[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
00594
                                                                 ctl->turb meso * vsiq);
00595
          atm->wp[ip] =
00596
            r * atm->wp[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
00597
                                                                 ctl->turb_meso * wsiq);
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 \rightarrow p[ip] += atm \rightarrow wp[ip] * dt;
00603
00604 }
00605
00607
00608 void module_diffusion_turb(
00609
        ctl_t * ctl,
00610
        atm_t * atm,
00611
        int ip,
00612
        double dt,
00613
        gsl_rng * rng) {
00614
00615
        double dx, dz, pt, p0, p1, w;
00616
00617
        /* Get tropopause pressure... */
00618
        pt = tropopause(atm->time[ip], atm->lat[ip]);
00619
        /* Get weighting factor... */
p1 = pt * 0.866877899;
p0 = pt / 0.866877899;
00620
00621
00622
00623
        if (atm->p[ip] > p0)
        w = 1;
else if (atm->p[ip] < p1)</pre>
00624
00625
00626
          w = 0;
00627
        else
00628
          w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00629
        /* Set diffusivitiy... */
00630
00631
        dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
        dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
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]);
          atm->lat[ip]
00639
00640
            += dy2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00641
                       / 1000.);
00642
00643
00644
        /* Vertical turbulent diffusion... */
        if (dz > 0)
00645
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,
        met_t * met0,
00655
        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
00666
        char line[LEN];
00667
        /\star Check control parameter... \star/
00668
        if (ctl->isosurf < 1 || ctl->isosurf > 4)
00669
00670
          return;
00671
00672
        /* Initialize... */
00673
        if (ip < 0) {</pre>
00674
00675
           /* Allocate... */
00676
          ALLOC(iso, double,
00677
                 NP);
00678
          ALLOC(ps, double,
00679
                nb);
00680
          ALLOC(ts, double,
00681
                nb);
00682
00683
          /* Save pressure... */
00684
          if (ctl->isosurf == 1)
00685
             for (ip2 = 0; ip2 < atm->np; ip2++)
00686
              iso[ip2] = atm->p[ip2];
00687
          /* Save density... */
else if (ctl->isosurf == 2)
00688
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);
               iso[ip2] = atm->p[ip2] / t;
00694
00695
00696
00697
           /\star Save potential temperature... \star/
00698
          else if (ctl->isosurf == 3)
00699
             for (ip2 = 0; ip2 < atm->np; ip2++) {
              intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00700
                                atm->lon[ip2], atm->lat[ip2], NULL, &t, NULL, NULL,
00701
00702
                                NULL, NULL, NULL);
00703
               iso[ip2] = t * pow(P0 / atm->p[ip2], 0.286);
00704
00705
00706
           /* Read balloon pressure data... */
00707
          else if (ctl->isosurf == 4) {
00708
00709
             /* Write info... */
00710
            printf("Read balloon pressure data: sn'', ctl->balloon);
00711
00712
             /* Open file... */
             if (!(in = fopen(ctl->balloon, "r")))
00713
               ERRMSG("Cannot open file!");
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
             /\star Check number of points... \star/
00723
             if (n < 1)
               ERRMSG("Could not read any data!");
00724
00725
00726
             /* Close file... */
00727
             fclose(in);
00728
00729
00730
           /* Leave initialization... */
00731
          return:
00732
00733
00734
        /* Restore pressure... */
00735
        if (ctl->isosurf == 1)
00736
          atm->p[ip] = iso[ip];
00737
00738
        /* Restore density... */
```

5.32 trac.c 175

```
else if (ctl->isosurf == 2) {
         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
        /\star Restore potential temperature... \star/
00746
       else if (ctl->isosurf == 3) {
00747
         intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
     lon[ip],
         atm->p[ip] = P0 * pow(iso[ip] / t, -1. / 0.286);
00748
00749
00750
00751
00752
        /* Interpolate pressure... */
       else if (ctl->isosurf == 4) {
  if (atm->time[ip] <= ts[0])</pre>
00753
00754
00755
           atm->p[ip] = ps[0];
         else if (atm->time[ip] >= ts[n - 1])
00756
00757
           atm->p[ip] = ps[n - 1];
00758
         else {
           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,
00770
       met_t * met0,
00771
       met_t * met1,
       atm_t * atm,
00772
00773
       int ip) {
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
                         00779
                         NULL, NULL, NULL, NULL, NULL);
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],
                         atm->lat[ip], NULL, &atm->q[ctl->qnt_t][ip],
00784
                         NULL, NULL, NULL, NULL, NULL);
00785
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)
         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->
     lon[ip],
00808
                         atm->lat[ip], NULL, NULL, NULL, NULL, NULL,
00809
                         &atm->q[ctl->qnt_h2o][ip], NULL);
00810
00811
        /* Interpolate ozone... */
00812
       if (ct1->qnt_o3>=0)
         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
                         \alpha = -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->
     lon[ip],
         00820
00821
00823
00824 }
00825
00827
00828 void module_position(
00829
       met_t * met0,
00830
       met_t * met1,
00831
       atm_t * atm,
00832
       int ip) {
00833
00834
       double ps;
00835
00836
        /* Calculate modulo... */
       atm->lon[ip] = fmod(atm->lon[ip], 360);
atm->lat[ip] = fmod(atm->lat[ip], 360);
00837
00838
00839
00840
        /* Check latitude... */
       while (atm->lat[ip] > -90 || atm->lat[ip] > 90) {
   if (atm->lat[ip] > 90) {
00841
00842
00843
           atm->lat[ip] = 180 - atm->lat[ip];
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
00852
       /* Check longitude... */
       while (atm->lon[ip] < -180)
00853
00854
         atm->lon[ip] += 360;
00855
       while (atm->lon[ip] >= 180)
00856
         atm->lon[ip] -= 360;
00857
00858
       /* Get surface pressure... */
intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00859
                        atm->lon[ip], atm->lat[ip], &ps, NULL,
00860
00861
                        NULL, NULL, NULL, NULL, NULL);
00862
00863
       /* Check pressure... */
       if (atm->p[ip] > ps)
00864
00865
       atm->p[ip] = ps;
else if (atm->p[ip] < met0->p[met0->np - 1])
00866
00867
         atm->p[ip] = met0->p[met0->np - 1];
00868 }
00869
00871
00872 void module_sedi(
00873
       ctl_t * ctl,
00874
       met_t * met0,
00875
       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
       /\star Specific gas constant for dry air [J/(kg K)]: \star/
00883
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)</pre>
00893
         return;
00894
00895
       /* Convert units... */
       p = 100 * atm->p[ip];
00896
00897
       r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
00898
       rho_p = atm->q[ctl->qnt_rho][ip];
00899
        /* Get temperature... */
00900
       intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00901
      lon[ip],
```

```
atm->lat[ip], NULL, &T, NULL, NULL, NULL, NULL, NULL);
00903
00904
        /* Density of dry air... */
00905
        rho = p / (R * T);
00906
        /* Dynamic viscosity of air... */ eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
00907
00909
        /* Thermal velocity of an air molecule... */
00910
        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 -
                                  rho) * GSL_CONST_MKSA_GRAV_ACCEL / (9. * eta) * G;
00925
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,
       met_t * met1,
atm_t * atm,
00937
00938
00939
       double t) {
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
00950
       /* Write atmospheric data... */
if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
00951
        sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
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) {
00958
00959
         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
00960
                  dirname, ctl->grid_basename, year, mon, day, hour, min);
00961
          write_grid(filename, ctl, met0, met1, atm, t);
00962
00963
00964
        /* Write CSI data... */
00965
        if (ctl->csi_basename[0] != '-') {
00966
         sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
00967
         write_csi(filename, ctl, atm, t);
00968
00969
00970
       /* Write profile data...
        if (ctl->prof_basename[0] != '-') {
        sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
00972
00973
          write_prof(filename, ctl, met0, met1, atm, t);
00974
00975
00976
        /* Write station data...
       /* Write station data... */
if (ctl->stat_basename[0] != '-') {
00978
        sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
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
             /* Read control parameters... */
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_LUT", -1, "38.587660177302", NULL);
alpha = scan_ctl(argv[1], argc, argv, "WIND_LUT", -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, "T", NC_FLOAT, 4, &dims[0], &tid));
NC(nc_def_var(ncid, "U", NC_FLOAT, 4, &dims[0], &uid));
NC(nc_def_var(ncid, "V", NC_FLOAT, 4, &dims[0], &vid));
00103
00104
00105
            NC(nc_def_var(ncid, "W", NC_FLOAT, 4, &dims[0], &wid));
00106
00107
           /* Set attributes... */
add_text_attribute(ncid, "time", "long_name", "time");
add_text_attribute(ncid, "time", "units", "day as %Y%m%d.%f");
add_text_attribute(ncid, "lon", "long_name", "longitude");
add_text_attribute(ncid, "lon", "units", "degrees_east");
add_text_attribute(ncid, "lat", "long_name", "latitude");
add_text_attribute(ncid, "lat", "units", "degrees_north");
add_text_attribute(ncid, "lev", "long_name", "air_pressure");
add_text_attribute(ncid, "lev", "units", "Pa");
add_text_attribute(ncid, "T", "long_name", "Temperature");
add_text_attribute(ncid, "T", "units", "K");
add_text_attribute(ncid, "U", "long_name", "U velocity");
add_text_attribute(ncid, "U", "units", "m s**-1");
add_text_attribute(ncid, "V", "long_name", "V velocity");
add_text_attribute(ncid, "V", "units", "m s**-1");
             /* Set attributes...
00108
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
00119
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", "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
             /* 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;

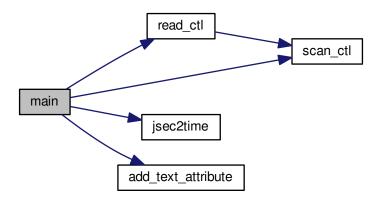
dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
00148
                                                            * (cos(dataLat[iy] * M_PI / 180.0)

* 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
00160
            NC(nc_put_var_float(ncid, tid, dataT));
00161
            NC(nc_put_var_float(ncid, uid, dataU));
00162
             NC(nc_put_var_float(ncid, vid, dataV));
            NC(nc_put_var_float(ncid, wid, dataW));
00163
00164
00165
             /* Close file... */
00166
            NC (nc_close (ncid));
00167
```

```
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
         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
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
          Main...
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],
```

5.34 wind.c 181

```
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,
                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
00064
             t0 = scan_ctl(argv[1], argc, argv, "WIND_TO", -1, "0", NULL);

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

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

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

20 = 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_UI", -1, "38.587660177302", NULL);

alpha = scan_ctl(argv[1], argc, argv, "WIND_UI", -1, "0.0", NULL);
             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>
             if (ny < 1 || ny > EY)
    ERRMSG("Set 1 <= NY <= MAX!");</pre>
00077
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", "long_name", "Temperature");
              /* 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
             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
00126
               /* End definition... */
00127
             NC(nc_enddef(ncid));
00128
00129
              /* Set coordinates... */
             for (ix = 0; ix < nx; ix++)
00130
                dataLon[ix] = 360.0 / nx * (double) ix;
00131
              for (iy = 0; iy < ny; iy++)
  dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;</pre>
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)... */
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 < nz; 1z++; 1

idx = (iz * ny + iy) * nx + ix;

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

* (cos(dataLat[iy] * M_PI / 180.0)

* cos(alpha * M_PI / 180.0)
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, 12
wind.c, 178	atm_dt_out, 12
atm basename	atm_gpfile, 12
ctl_t, 12	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, 14
lat, 4	csi_lon0, 13
lon, 4	csi_lon1, 13
np, 4	csi_modmin, 13
p, 4	csi_nx, 13
φ, 4 q, 4	csi_ny, 13
time, 4	csi_nz, 13
up, 4	csi_obsfile, 12
vp, 4	csi_obsmin, 12
wp, 5	csi_z0, 13
<b>Ψρ</b> , <b>3</b>	csi_z1, 13
balloon	direction, 10
ctl_t, 11	dt_met, 10
<b>-</b> '	dt_mod, 10
cart2geo	grid_basename, 14
libtrac.c, 39	grid_dt_out, 14
libtrac.h, 89	grid_gpfile, 14
center.c, 20	grid_lat0, 15
main, 20	grid_lat1, 15
csi_basename	grid_lon0, 15 grid_lon1, 15
ctl_t, 12	grid_nx, 14
csi_dt_out	grid_ny, 15
ctl_t, 12	grid_nz, 14
csi_lat0	grid_sparse, 14
ctl_t, 13	grid_sparse, 14 grid z0, 14
csi_lat1	grid_20, 14 grid_z1, 14
ctl_t, 14	isosurf, 11
csi_lon0	met_np, 11
ctl_t, 13	met_p, 11
csi_lon1	nq, 8
ctl_t, 13	prof_basename, 15
csi_modmin	prof_lat0, 16
ctl_t, 13	prof_lat1, 16
csi_nx ctl_t, 13	prof_lon0, 16
	prof_lon1, 16
csi_ny	prof_nx, 16
ctl_t, 13 csi_nz	prof_ny, 16
ctl_t, 13	prof nz, 15
csi_obsfile	prof_obsfile, 15
ctl_t, 12	prof_z0, 15
csi_obsmin	prof_z1, 16
ctl_t, 12	qnt_format, 9
csi_z0	qnt_h2o, 10
ctl_t, 13	qnt_m, 9
csi_z1	qnt_name, 8
ctl_t, 13	qnt o3, 10
otl t 5	ant ne 0

qnt\_ps, 9

qnt_r, 9	get_met_help
qnt_rho, 9	libtrac.c, 42
qnt_stat, 10	libtrac.h, 91
qnt_t, 9	grid_basename
qnt_theta, 10	ctl_t, 14
qnt_u, 9	grid_dt_out
qnt_unit, 8	ctl_t, 14
qnt_v, 9	grid_gpfile
qnt_w, 9	ctl_t, 14
stat_basename, 16	grid_lat0
stat_lat, 17	ctl_t, 15
stat_lon, 16	grid_lat1
stat_r, 17	ctl_t, 15
t_start, 10	grid_lon0
t_stop, 10	ctl_t, 15
tdec_strat, 12	grid_lon1
tdec_trop, 12	ctl_t, 15
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, 15
turb_meso, 11	grid_nz
turb_meso, 11	-
deg2dx	ctl_t, 14
libtrac.c, 39	grid_sparse
libtrac.h, 89	ctl_t, 14
deg2dy	grid_z0
libtrac.c, 39	ctl_t, 14
	grid_z1
libtrac.h, 89	ctl_t, 14
direction	
ctl_t, 10 dist.c, 24	h2o
niet c 24	
	met_t, 19
main, 24	
main, 24 dp2dz	init.c, 32
main, 24 dp2dz libtrac.c, 40	init.c, 32 main, 33
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89	init.c, 32 main, 33 init_simtime
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met	init.c, 32 main, 33 init_simtime trac.c, 153
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod ctl_t, 10	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92 intpol_met_3d
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92 intpol_met_3d libtrac.c, 43
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92 intpol_met_3d libtrac.c, 43 libtrac.h, 92
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92 intpol_met_3d libtrac.c, 43 libtrac.h, 92 intpol_met_space
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 90	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92 intpol_met_3d libtrac.c, 43 libtrac.h, 92 intpol_met_space libtrac.c, 43
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 90 dy2deg	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92 intpol_met_3d libtrac.c, 43 libtrac.h, 92 intpol_met_space libtrac.c, 43 libtrac.h, 93
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 90 dy2deg libtrac.c, 40	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92 intpol_met_3d libtrac.c, 43 libtrac.h, 92 intpol_met_space libtrac.c, 43
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 90 dy2deg libtrac.c, 40 libtrac.h, 90	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92 intpol_met_3d libtrac.c, 43 libtrac.h, 92 intpol_met_space libtrac.c, 43 libtrac.h, 93
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 90 dy2deg libtrac.c, 40 libtrac.h, 90 dz2dp	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92 intpol_met_3d libtrac.c, 43 libtrac.h, 92 intpol_met_space libtrac.c, 43 libtrac.h, 93 intpol_met_time
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 90 dy2deg libtrac.c, 40 libtrac.h, 90 dz2dp libtrac.c, 40	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92 intpol_met_3d libtrac.c, 43 libtrac.h, 92 intpol_met_space libtrac.c, 43 libtrac.h, 93 intpol_met_time libtrac.c, 44
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 90 dy2deg libtrac.c, 40 libtrac.h, 90 dz2dp libtrac.c, 40	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92 intpol_met_3d libtrac.c, 43 libtrac.h, 92 intpol_met_space libtrac.c, 43 libtrac.h, 93 intpol_met_time libtrac.c, 44 libtrac.h, 94
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 90 dy2deg libtrac.c, 40 libtrac.h, 90 dz2dp libtrac.c, 40 libtrac.h, 90 dz2dp libtrac.c, 40 libtrac.h, 90	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92 intpol_met_3d libtrac.c, 43 libtrac.h, 92 intpol_met_space libtrac.c, 43 libtrac.h, 93 intpol_met_time libtrac.c, 44 libtrac.h, 94 isosurf
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 90 dy2deg libtrac.c, 40 libtrac.h, 90 dz2dp libtrac.c, 40 libtrac.h, 90 extract.c, 30	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92 intpol_met_3d libtrac.c, 43 libtrac.h, 92 intpol_met_space libtrac.c, 43 libtrac.h, 93 intpol_met_time libtrac.c, 44 libtrac.h, 94 isosurf
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 90 dy2deg libtrac.c, 40 libtrac.h, 90 dz2dp libtrac.c, 40 libtrac.h, 90 extract.c, 30 main, 30 geo2cart	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92 intpol_met_3d libtrac.c, 43 libtrac.h, 92 intpol_met_space libtrac.c, 43 libtrac.h, 93 intpol_met_time libtrac.c, 44 libtrac.h, 94 isosurf ctl_t, 11
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 90 dy2deg libtrac.c, 40 libtrac.h, 90 dz2dp libtrac.c, 40 libtrac.h, 90 extract.c, 30 main, 30	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92 intpol_met_3d libtrac.c, 43 libtrac.h, 92 intpol_met_space libtrac.c, 43 libtrac.h, 93 intpol_met_time libtrac.c, 44 libtrac.h, 94 isosurf ctl_t, 11
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 90 dy2deg libtrac.c, 40 libtrac.h, 90 dz2dp libtrac.c, 40 libtrac.h, 90 extract.c, 30 main, 30 geo2cart	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92 intpol_met_3d libtrac.c, 43 libtrac.h, 92 intpol_met_space libtrac.c, 43 libtrac.h, 93 intpol_met_time libtrac.c, 44 libtrac.h, 94 isosurf ctl_t, 11  jsec2time libtrac.c, 45
main, 24 dp2dz     libtrac.c, 40     libtrac.h, 89 dt_met     ctl_t, 10 dt_mod     ctl_t, 10 dx2deg     libtrac.c, 40     libtrac.h, 90 dy2deg     libtrac.c, 40     libtrac.h, 90 dz2dp     libtrac.c, 40     libtrac.h, 90 extract.c, 30     main, 30  geo2cart     libtrac.c, 40	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92 intpol_met_3d libtrac.c, 43 libtrac.h, 92 intpol_met_space libtrac.c, 43 libtrac.h, 93 intpol_met_time libtrac.c, 44 libtrac.h, 94 isosurf ctl_t, 11  jsec2time libtrac.c, 45 libtrac.h, 95 jsec2time.c, 36
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 90 dy2deg libtrac.c, 40 libtrac.h, 90 dz2dp libtrac.c, 40 libtrac.h, 90 extract.c, 30 main, 30  geo2cart libtrac.h, 90	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92 intpol_met_3d libtrac.c, 43 libtrac.h, 92 intpol_met_space libtrac.c, 43 libtrac.h, 93 intpol_met_time libtrac.c, 44 libtrac.h, 94 isosurf ctl_t, 11  jsec2time libtrac.c, 45 libtrac.h, 95
main, 24 dp2dz libtrac.c, 40 libtrac.h, 89 dt_met ctl_t, 10 dt_mod ctl_t, 10 dx2deg libtrac.c, 40 libtrac.h, 90 dy2deg libtrac.c, 40 libtrac.h, 90 dz2dp libtrac.c, 40 libtrac.h, 90 extract.c, 30 main, 30  geo2cart libtrac.h, 90 get_met	init.c, 32 main, 33 init_simtime trac.c, 153 intpol_met_2d libtrac.c, 42 libtrac.h, 92 intpol_met_3d libtrac.c, 43 libtrac.h, 92 intpol_met_space libtrac.c, 43 libtrac.h, 93 intpol_met_time libtrac.c, 44 libtrac.h, 94 isosurf ctl_t, 11  jsec2time libtrac.c, 45 libtrac.h, 95 jsec2time.c, 36

atm_t, 4	read_met_periodic, 103
met_t, 18	scan_ctl, 103
libtrac.c, 37	time2jsec, 104
cart2geo, 39	timer, 104
deg2dx, 39	tropopause, 105
deg2dy, 39	write_atm, 107
dp2dz, 40	write_csi, 108
dx2deg, 40	write_grid, 110
dy2deg, 40	write_prof, 112
dz2dp, 40	write_station, 114
geo2cart, 40	locate
get_met, 41	libtrac.c, 45
get_met_help, 42	libtrac.h, 95
intpol_met_2d, 42	lon
intpol_met_3d, 43	atm_t, 4
intpol_met_space, 43	met_t, 18
intpol_met_time, 44	main
jsec2time, 45	main
locate, 45	center.c, 20
read_atm, 46	dist.c, 24
read ctl, 46	extract.c, 30
read_met, 49	init.c, 33
read_met_extrapolate, 51	jsec2time.c, 36
read_met_help, 52	match.c, 123
read_met_ml2pl, 52	met_map.c, 127
read_met_periodic, 53	met_prof.c, 131
scan_ctl, 53	met_sample.c, 136
time2jsec, 54	met_zm.c, 139
timer, 54	smago.c, 143
tropopause, 55	split.c, 147
write_atm, 57	time2jsec.c, 151
	trac.c, 163
write_csi, 58	wind.c, 178
write_grid, 60	match.c, 122
write_prof, 62	main, 123
write_station, 64	met_map.c, 127
libtrac.h, 87	main, 127
cart2geo, 89	met_np
deg2dx, 89	ctl_t, 11
deg2dy, 89	met_p
dp2dz, 89	ctl_t, 11
dx2deg, 90	met_prof.c, 131
dy2deg, 90	main, 131
dz2dp, 90	met_sample.c, 135
geo2cart, 90	main, 136
get_met, 90	met_t, 17
get_met_help, 91	h2o, 19
intpol_met_2d, 92	lat, 18
intpol_met_3d, 92	lon, 18
intpol_met_space, 93	np, 18
intpol_met_time, 94	nx, 18
jsec2time, 95	ny, 18
locate, 95	o3, 19
read_atm, 96	p, 18
read_ctl, 96	pl, 19
read_met, 99	ps, 18
read_met_extrapolate, 101	t, 19
read_met_help, 101	time, 18
read_met_ml2pl, 102	u, 19
	-,

v, 19	prof_z1
w, 19	ctl_t, 16
met_zm.c, 138	ps
main, 139	met_t, 18
module_advection	
trac.c, 153	q
module_decay	atm_t, 4
trac.c, 153	qnt_format
module_diffusion_meso	ctl_t, 9
trac.c, 154	qnt_h2o
module_diffusion_turb	ctl_t, 10
trac.c, 156	qnt_m
module_isosurf	ctl_t, 9
trac.c, 157	qnt_name
module_meteo	ctl_t, 8 qnt_o3
trac.c, 159	ctl_t, 10
module_position	gnt ps
trac.c, 160	ctl_t, 9
module_sedi	qnt_r
trac.c, 161	ctl_t, 9
nn	qnt_rho
np atm_t, 4	ctl_t, 9
met_t, 18	qnt_stat
nq	ctl_t, 10
ctl_t, 8	qnt_t
nx	ctl_t, 9
met_t, 18	qnt_theta
ny	ctl_t, 10
met_t, 18	qnt_u
	ctl_t, 9
03	qnt_unit
met_t, 19	ctl_t, 8
	qnt_v
p	ctl_t, 9
atm_t, 4	qnt_w
met_t, 18	ctl_t, 9
pl	
met_t, 19	read_atm
prof_basename	libtrac.c, 46
ctl_t, 15	libtrac.h, 96
prof_lat0	read_ctl
ctl_t, 16	libtrac.c, 46
prof_lat1	libtrac.h, 96
ctl_t, 16	read_met
prof_lon0	libtrac.c, 49
ctl_t, 16	libtrac.h, 99
prof_lon1	read_met_extrapolate
ctl_t, 16	libtrac.c, 51
prof_nx	libtrac.h, 101
ctl_t, 16	read_met_help
prof_ny	libtrac.c, 52
ctl_t, 16	libtrac.h, 101
prof_nz	read_met_ml2pl
ctl_t, 15	libtrac.c, 52
prof_obsfile	libtrac.h, 102
ctl_t, 15	read_met_periodic
prof_z0	libtrac.c, 53
ctl_t, 15	libtrac.h, 103

scan_ctl	ctl_t, 11
libtrac.c, 53	turb_meso
libtrac.h, 103	_ ctl_t, 11
smago.c, 143	_ <i>,</i>
main, 143	u
split.c, 146	met_t, 19
•	up
main, 147	atm_t, 4
stat_basename	atm_t, <del>-</del>
ctl_t, 16	V
stat_lat	met t, 19
ctl_t, 17	<del>-</del> '
stat_lon	vp
ctl_t, 16	atm_t, 4
stat_r	14/
ctl_t, 17	W mot + 10
	met_t, 19
t	wind.c, 177
met_t, 19	add_text_attribute, 178
t_start	main, 178
ctl_t, 10	wp
t stop	atm_t, 5
ctl t, 10	write_atm
tdec_strat	libtrac.c, 57
ctl_t, 12	libtrac.h, 107
tdec_trop	write_csi
_ ·	libtrac.c, 58
ctl_t, 12	libtrac.h, 108
time	write_grid
atm_t, 4	— <del>-</del>
met_t, 18	libtrac.c, 60
time2jsec	libtrac.h, 110
libtrac.c, 54	write_output
libtrac.h, 104	trac.c, 162
time2jsec.c, 150	write_prof
main, 151	libtrac.c, 62
timer	libtrac.h, 112
libtrac.c, 54	write_station
libtrac.h, 104	libtrac.c, 64
trac.c, 152	libtrac.h, 114
init_simtime, 153	
main, 163	
module_advection, 153	
module_decay, 153	
module_diffusion_meso, 154	
module_diffusion_turb, 156	
module_isosurf, 157	
module_meteo, 159	
module_position, 160	
module_sedi, 161	
write_output, 162	
tropopause	
libtrac.c, 55	
libtrac.h, 105	
turb_dx_strat	
ctl_t, 11	
turb_dx_trop	
ctl_t, 11	
turb_dz_strat	
ctl_t, 11	
turb_dz_trop	