MPTRAC

Generated by Doxygen 1.8.11

ii CONTENTS

Contents

1	Main	n Page	2
2	Data	Structure Index	2
	2.1	Data Structures	2
3	File	Index	2
	3.1	File List	2
4	Data	Structure Documentation	3
	4.1	atm_t Struct Reference	3
		4.1.1 Detailed Description	4
		4.1.2 Field Documentation	4
	4.2	ctl_t Struct Reference	5
		4.2.1 Detailed Description	8
		4.2.2 Field Documentation	8
	4.3	met_t Struct Reference	18
		4.3.1 Detailed Description	19
		4.3.2 Field Documentation	19
5	File	Documentation	21
	5.1	center.c File Reference	21
		5.1.1 Detailed Description	21
		5.1.2 Function Documentation	21
	5.2	center.c	23
	5.3	dist.c File Reference	25
		5.3.1 Detailed Description	25
		5.3.2 Function Documentation	25
	5.4	dist.c	28
	5.5	extract.c File Reference	31
		5.5.1 Detailed Description	31
		5.5.2 Function Documentation	31

5.6	extract.c	32
5.7	init.c File Reference	33
	5.7.1 Detailed Description	34
	5.7.2 Function Documentation	34
5.8	init.c	35
5.9	jsec2time.c File Reference	37
	5.9.1 Detailed Description	37
	5.9.2 Function Documentation	37
5.10	jsec2time.c	38
5.11	libtrac.c File Reference	38
	5.11.1 Detailed Description	40
	5.11.2 Function Documentation	40
5.12	libtrac.c	67
5.13	libtrac.h File Reference	89
	5.13.1 Detailed Description	90
	5.13.2 Function Documentation	90
5.14	libtrac.h	117
	match.c File Reference	
	5.15.1 Detailed Description	
	5.15.2 Function Documentation	
5.16	match.c	
	met_map.c File Reference	
	5.17.1 Detailed Description	
	5.17.2 Function Documentation	
5 18	met map.c	
	met_prof.c File Reference	
5.19		
	5.19.1 Detailed Description	
F.0-	5.19.2 Function Documentation	
	met_prof.c	
5.21	met_sample.c File Reference	137

	5.21.1 Detailed Description	137
	5.21.2 Function Documentation	137
5.22	met_sample.c	139
5.23	met_zm.c File Reference	140
	5.23.1 Detailed Description	140
	5.23.2 Function Documentation	141
5.24	met_zm.c	143
5.25	smago.c File Reference	145
	5.25.1 Detailed Description	145
	5.25.2 Function Documentation	145
5.26	smago.c	147
5.27	split.c File Reference	148
	5.27.1 Detailed Description	148
	5.27.2 Function Documentation	149
5.28	split.c	151
5.29	time2jsec.c File Reference	152
	5.29.1 Detailed Description	152
	5.29.2 Function Documentation	153
5.30	time2jsec.c	153
5.31	trac.c File Reference	154
	5.31.1 Detailed Description	154
	5.31.2 Function Documentation	155
5.32	trac.c	169
5.33	wind.c File Reference	181
	5.33.1 Detailed Description	181
	5.33.2 Function Documentation	181
5.34	wind.c	183

187

Index

1 Main Page

Massive-Parallel Trajectory Calculations (MPTRAC) is a Lagrangian particle dispersion model for the troposphere and stratosphere. This reference manual provides information on the algorithms and data structures used in the code. Further information can be found at: http://www.fz-juelich.de/ias/jsc/mptrac

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

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

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

center.c Calculate center of mass of air parcels	21
dist.c Calculate transport deviations of trajectories	25
extract.c Extract single trajectory from atmospheric data files	31
init.c Create atmospheric data file with initial air parcel positions	33
jsec2time.c Convert Julian seconds to date	37
libtrac.c MPTRAC library definitions	38
libtrac.h MPTRAC library declarations	89
match.c Calculate deviations between two trajectories	124

met_map.c Extract global map from meteorological data	128
met_prof.c Extract vertical profile from meteorological data	132
met_sample.c Sample meteorological data at given geolocations	137
met_zm.c Extract zonal mean from meteorological data	140
smago.c Estimate horizontal diffusivity based on Smagorinsky theory	145
split.c Split air parcels into a larger number of parcels	148
time2jsec.c Convert date to Julian seconds	152
trac.c Lagrangian particle dispersion model	154
wind.c Create meteorological data files with synthetic wind fields	181
4 Data Structure Documentation	
4.1 atm_t Struct Reference	
Atmospheric data.	
<pre>#include <libtrac.h></libtrac.h></pre>	
Data Fields	
• int np	
Number of air pacels. • double time [NP]	
Time [s].	
• double p [NP]	
Pressure [hPa].	
double lon [NP]	
Longitude [deg]. • double lat [NP]	
Latitude [deg].	
double q [NQ][NP] Quantitiy data (for various, user-defined attributes). double up [ND]	
double up [NP] Zonal wind perturbation [m/s].	
double vp [NP]	
Meridional wind perturbation [m/s]. • double wp [NP]	

Vertical velocity perturbation [hPa/s].

```
4.1.1 Detailed Description
Atmospheric data.
Definition at line 425 of file libtrac.h.
4.1.2 Field Documentation
4.1.2.1 int atm_t::np
Number of air pacels.
Definition at line 428 of file libtrac.h.
4.1.2.2 double atm_t::time[NP]
Time [s].
Definition at line 431 of file libtrac.h.
4.1.2.3 double atm_t::p[NP]
Pressure [hPa].
Definition at line 434 of file libtrac.h.
4.1.2.4 double atm_t::lon[NP]
Longitude [deg].
Definition at line 437 of file libtrac.h.
4.1.2.5 double atm_t::lat[NP]
Latitude [deg].
Definition at line 440 of file libtrac.h.
4.1.2.6 double atm_t::q[NQ][NP]
Quantitiy data (for various, user-defined attributes).
Definition at line 443 of file libtrac.h.
4.1.2.7 double atm_t::up[NP]
Zonal wind perturbation [m/s].
Definition at line 446 of file libtrac.h.
```

```
4.1.2.8 double atm_t::vp[NP]
Meridional wind perturbation [m/s].
Definition at line 449 of file libtrac.h.
4.1.2.9 double atm_t::wp[NP]
Vertical velocity perturbation [hPa/s].
Definition at line 452 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_p
          Quantity array index for pressure.
    int qnt_t
          Quantity array index for temperature.
    • int qnt_u
          Quantity array index for zonal wind.
    int qnt_v
          Quantity array index for meridional wind.
    • int qnt w
```

Quantity array index for vertical velocity.

· int qnt_h2o Quantity array index for water vapor vmr. • int qnt o3 Quantity array index for ozone vmr. int qnt_theta Quantity array index for potential temperature. · int qnt pv Quantity array index for potential vorticity. int qnt_tice Quantity array index for T_ice. · int qnt_tnat Quantity array index for T_NAT. int qnt_stat Quantity array index for station flag. · int direction Direction flag (1=forward calculation, -1=backward calculation). double t_start Start time of simulation [s]. double t_stop Stop time of simulation [s]. · double dt_mod Time step of simulation [s]. · double dt met Time step of meteorological data [s]. int met_np Number of target pressure levels. double met_p [EP] Target pressure levels [hPa]. · int isosurf Isosurface parameter (0=none, 1=pressure, 2=density, 3=theta, 4=balloon). • char balloon [LEN] Balloon position filename. double turb_dx_trop Horizontal turbulent diffusion coefficient (troposphere) $[m^2/2]$. · double turb dx strat Horizontal turbulent diffusion coefficient (stratosphere) [m^2/s]. double turb dz trop Vertical turbulent diffusion coefficient (troposphere) $[m^2/s]$. double turb_dz_strat Vertical turbulent diffusion coefficient (stratosphere) $[m^2/s]$. · double turb meso Scaling factor for mesoscale wind fluctuations. · double tdec_trop Life time of particles (troposphere) [s]. · double tdec strat Life time of particles (stratosphere) [s]. • char atm_basename [LEN] Basename of atmospheric data files. char atm gpfile [LEN]

Gnuplot file for atmospheric data.

double atm_dt_out

Time step for atmospheric data output [s].

· int atm_filter

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

· char csi basename [LEN]

Basename of CSI data files.

· double csi dt out

Time step for CSI data output [s].

char csi obsfile [LEN]

Observation data file for CSI analysis.

· double csi_obsmin

Minimum observation index to trigger detection.

• double csi_modmin

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

• int csi_nz

Number of altitudes of gridded CSI data.

double csi z0

Lower altitude of gridded CSI data [km].

double csi_z1

Upper altitude of gridded CSI data [km].

· int csi nx

Number of longitudes of gridded CSI data.

• double csi_lon0

Lower longitude of gridded CSI data [deg].

double csi_lon1

Upper longitude of gridded CSI data [deg].

• int csi_ny

Number of latitudes of gridded CSI data.

double csi_lat0

Lower latitude of gridded CSI data [deg].

· double csi lat1

Upper latitude of gridded CSI data [deg].

char grid_basename [LEN]

Basename of grid data files.

char grid_gpfile [LEN]

Gnuplot file for gridded data.

· double grid_dt_out

Time step for gridded data output [s].

• int grid_sparse

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

• int grid_nz

Number of altitudes of gridded data.

• double grid_z0

Lower altitude of gridded data [km].

double grid_z1

Upper altitude of gridded data [km].

• int grid_nx

Number of longitudes of gridded data.

double grid_lon0

Lower longitude of gridded data [deg].

double grid_lon1

Upper longitude of gridded data [deg].

• int grid_ny

Number of latitudes of gridded data.

· double grid lat0

Lower latitude of gridded data [deg].

double grid_lat1

Upper latitude of gridded data [deg].

char prof basename [LEN]

Basename for profile output file.

char prof_obsfile [LEN]

Observation data file for profile output.

• int prof_nz

Number of altitudes of gridded profile data.

• double prof_z0

Lower altitude of gridded profile data [km].

double prof_z1

Upper altitude of gridded profile data [km].

int prof_nx

Number of longitudes of gridded profile data.

double prof lon0

Lower longitude of gridded profile data [deg].

double prof_lon1

Upper longitude of gridded profile data [deg].

• int prof_ny

Number of latitudes of gridded profile data.

double prof_lat0

Lower latitude of gridded profile data [deg].

double prof lat1

Upper latitude of gridded profile data [deg].

• char stat_basename [LEN]

Basename of station data file.

• double stat_lon

Longitude of station [deg].

double stat_lat

Latitude of station [deg].

double stat_r

Search radius around station [km].

4.2.1 Detailed Description

Control parameters.

Definition at line 173 of file libtrac.h.

4.2.2 Field Documentation

4.2.2.1 int ctl_t::nq

Number of quantities.

Definition at line 176 of file libtrac.h.

4.2.2.2 char ctl_t::qnt_name[NQ][LEN] Quantity names. Definition at line 179 of file libtrac.h. 4.2.2.3 char ctl_t::qnt_unit[NQ][LEN] Quantity units. Definition at line 182 of file libtrac.h. 4.2.2.4 char ctl_t::qnt_format[NQ][LEN] Quantity output format. Definition at line 185 of file libtrac.h. 4.2.2.5 int ctl_t::qnt_m Quantity array index for mass. Definition at line 188 of file libtrac.h. 4.2.2.6 int ctl_t::qnt_rho Quantity array index for particle density. Definition at line 191 of file libtrac.h. 4.2.2.7 int ctl_t::gnt_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_p Quantity array index for pressure. Definition at line 200 of file libtrac.h. 4.2.2.10 int ctl_t::qnt_t Quantity array index for temperature. Definition at line 203 of file libtrac.h.

```
4.2.2.11 int ctl_t::qnt_u
Quantity array index for zonal wind.
Definition at line 206 of file libtrac.h.
4.2.2.12 int ctl_t::qnt_v
Quantity array index for meridional wind.
Definition at line 209 of file libtrac.h.
4.2.2.13 int ctl_t::qnt_w
Quantity array index for vertical velocity.
Definition at line 212 of file libtrac.h.
4.2.2.14 int ctl_t::qnt_h2o
Quantity array index for water vapor vmr.
Definition at line 215 of file libtrac.h.
4.2.2.15 int ctl_t::qnt_o3
Quantity array index for ozone vmr.
Definition at line 218 of file libtrac.h.
4.2.2.16 int ctl_t::qnt_theta
Quantity array index for potential temperature.
Definition at line 221 of file libtrac.h.
4.2.2.17 int ctl_t::qnt_pv
Quantity array index for potential vorticity.
Definition at line 224 of file libtrac.h.
4.2.2.18 int ctl_t::qnt_tice
Quantity array index for T_ice.
Definition at line 227 of file libtrac.h.
4.2.2.19 int ctl_t::qnt_tnat
Quantity array index for T_NAT.
Definition at line 230 of file libtrac.h.
```

```
4.2.2.20 int ctl_t::qnt_stat
Quantity array index for station flag.
Definition at line 233 of file libtrac.h.
4.2.2.21 int ctl_t::direction
Direction flag (1=forward calculation, -1=backward calculation).
Definition at line 236 of file libtrac.h.
4.2.2.22 double ctl_t::t_start
Start time of simulation [s].
Definition at line 239 of file libtrac.h.
4.2.2.23 double ctl_t::t_stop
Stop time of simulation [s].
Definition at line 242 of file libtrac.h.
4.2.2.24 double ctl_t::dt_mod
Time step of simulation [s].
Definition at line 245 of file libtrac.h.
4.2.2.25 double ctl_t::dt_met
Time step of meteorological data [s].
Definition at line 248 of file libtrac.h.
4.2.2.26 int ctl_t::met_np
Number of target pressure levels.
Definition at line 251 of file libtrac.h.
4.2.2.27 double ctl_t::met_p[EP]
Target pressure levels [hPa].
Definition at line 254 of file libtrac.h.
4.2.2.28 int ctl_t::isosurf
Isosurface parameter (0=none, 1=pressure, 2=density, 3=theta, 4=balloon).
Definition at line 258 of file libtrac.h.
```

```
4.2.2.29 char ctl_t::balloon[LEN]
Balloon position filename.
Definition at line 261 of file libtrac.h.
4.2.2.30 double ctl_t::turb_dx_trop
Horizontal turbulent diffusion coefficient (troposphere) [m<sup>2</sup>/s].
Definition at line 264 of file libtrac.h.
4.2.2.31 double ctl_t::turb_dx_strat
Horizontal turbulent diffusion coefficient (stratosphere) [m<sup>2</sup>/s].
Definition at line 267 of file libtrac.h.
4.2.2.32 double ctl_t::turb_dz_trop
Vertical turbulent diffusion coefficient (troposphere) [m<sup>2</sup>/s].
Definition at line 270 of file libtrac.h.
4.2.2.33 double ctl_t::turb_dz_strat
Vertical turbulent diffusion coefficient (stratosphere) [m<sup>2</sup>/s].
Definition at line 273 of file libtrac.h.
4.2.2.34 double ctl_t::turb_meso
Scaling factor for mesoscale wind fluctuations.
Definition at line 276 of file libtrac.h.
4.2.2.35 double ctl_t::tdec_trop
Life time of particles (troposphere) [s].
Definition at line 279 of file libtrac.h.
4.2.2.36 double ctl_t::tdec_strat
Life time of particles (stratosphere) [s].
Definition at line 282 of file libtrac.h.
4.2.2.37 char ctl_t::atm_basename[LEN]
Basename of atmospheric data files.
Definition at line 285 of file libtrac.h.
```

```
4.2.2.38 char ctl_t::atm_gpfile[LEN]
Gnuplot file for atmospheric data.
Definition at line 288 of file libtrac.h.
4.2.2.39 double ctl_t::atm_dt_out
Time step for atmospheric data output [s].
Definition at line 291 of file libtrac.h.
4.2.2.40 int ctl_t::atm_filter
Time filter for atmospheric data output (0=no, 1=yes).
Definition at line 294 of file libtrac.h.
4.2.2.41 char ctl_t::csi_basename[LEN]
Basename of CSI data files.
Definition at line 297 of file libtrac.h.
4.2.2.42 double ctl_t::csi_dt_out
Time step for CSI data output [s].
Definition at line 300 of file libtrac.h.
4.2.2.43 char ctl_t::csi_obsfile[LEN]
Observation data file for CSI analysis.
Definition at line 303 of file libtrac.h.
4.2.2.44 double ctl_t::csi_obsmin
Minimum observation index to trigger detection.
Definition at line 306 of file libtrac.h.
4.2.2.45 double ctl_t::csi_modmin
Minimum column density to trigger detection [kg/m<sup>2</sup>].
Definition at line 309 of file libtrac.h.
4.2.2.46 int ctl_t::csi_nz
Number of altitudes of gridded CSI data.
Definition at line 312 of file libtrac.h.
```

```
4.2.2.47 double ctl_t::csi_z0
Lower altitude of gridded CSI data [km].
Definition at line 315 of file libtrac.h.
4.2.2.48 double ctl_t::csi_z1
Upper altitude of gridded CSI data [km].
Definition at line 318 of file libtrac.h.
4.2.2.49 int ctl_t::csi_nx
Number of longitudes of gridded CSI data.
Definition at line 321 of file libtrac.h.
4.2.2.50 double ctl_t::csi_lon0
Lower longitude of gridded CSI data [deg].
Definition at line 324 of file libtrac.h.
4.2.2.51 double ctl_t::csi_lon1
Upper longitude of gridded CSI data [deg].
Definition at line 327 of file libtrac.h.
4.2.2.52 int ctl_t::csi_ny
Number of latitudes of gridded CSI data.
Definition at line 330 of file libtrac.h.
4.2.2.53 double ctl_t::csi_lat0
Lower latitude of gridded CSI data [deg].
Definition at line 333 of file libtrac.h.
4.2.2.54 double ctl_t::csi_lat1
Upper latitude of gridded CSI data [deg].
Definition at line 336 of file libtrac.h.
4.2.2.55 char ctl_t::grid_basename[LEN]
Basename of grid data files.
```

Definition at line 339 of file libtrac.h.

```
4.2.2.56 char ctl_t::grid_gpfile[LEN]
Gnuplot file for gridded data.
Definition at line 342 of file libtrac.h.
4.2.2.57 double ctl_t::grid_dt_out
Time step for gridded data output [s].
Definition at line 345 of file libtrac.h.
4.2.2.58 int ctl_t::grid_sparse
Sparse output in grid data files (0=no, 1=yes).
Definition at line 348 of file libtrac.h.
4.2.2.59 int ctl_t::grid_nz
Number of altitudes of gridded data.
Definition at line 351 of file libtrac.h.
4.2.2.60 double ctl_t::grid_z0
Lower altitude of gridded data [km].
Definition at line 354 of file libtrac.h.
4.2.2.61 double ctl_t::grid_z1
Upper altitude of gridded data [km].
Definition at line 357 of file libtrac.h.
4.2.2.62 int ctl_t::grid_nx
Number of longitudes of gridded data.
Definition at line 360 of file libtrac.h.
4.2.2.63 double ctl_t::grid_lon0
Lower longitude of gridded data [deg].
Definition at line 363 of file libtrac.h.
4.2.2.64 double ctl_t::grid_lon1
Upper longitude of gridded data [deg].
Definition at line 366 of file libtrac.h.
```

```
4.2.2.65 int ctl_t::grid_ny
Number of latitudes of gridded data.
Definition at line 369 of file libtrac.h.
4.2.2.66 double ctl_t::grid_lat0
Lower latitude of gridded data [deg].
Definition at line 372 of file libtrac.h.
4.2.2.67 double ctl_t::grid_lat1
Upper latitude of gridded data [deg].
Definition at line 375 of file libtrac.h.
4.2.2.68 char ctl_t::prof_basename[LEN]
Basename for profile output file.
Definition at line 378 of file libtrac.h.
4.2.2.69 char ctl_t::prof_obsfile[LEN]
Observation data file for profile output.
Definition at line 381 of file libtrac.h.
4.2.2.70 int ctl_t::prof_nz
Number of altitudes of gridded profile data.
Definition at line 384 of file libtrac.h.
4.2.2.71 double ctl_t::prof_z0
Lower altitude of gridded profile data [km].
Definition at line 387 of file libtrac.h.
4.2.2.72 double ctl_t::prof_z1
Upper altitude of gridded profile data [km].
Definition at line 390 of file libtrac.h.
4.2.2.73 int ctl_t::prof_nx
Number of longitudes of gridded profile data.
Definition at line 393 of file libtrac.h.
```

4.2.2.74 double ctl_t::prof_lon0 Lower longitude of gridded profile data [deg]. Definition at line 396 of file libtrac.h. 4.2.2.75 double ctl_t::prof_lon1 Upper longitude of gridded profile data [deg]. Definition at line 399 of file libtrac.h. 4.2.2.76 int ctl_t::prof_ny Number of latitudes of gridded profile data. Definition at line 402 of file libtrac.h. 4.2.2.77 double ctl_t::prof_lat0 Lower latitude of gridded profile data [deg]. Definition at line 405 of file libtrac.h. 4.2.2.78 double ctl_t::prof_lat1 Upper latitude of gridded profile data [deg]. Definition at line 408 of file libtrac.h. 4.2.2.79 char ctl_t::stat_basename[LEN] Basename of station data file. Definition at line 411 of file libtrac.h. 4.2.2.80 double ctl_t::stat_lon Longitude of station [deg]. Definition at line 414 of file libtrac.h. 4.2.2.81 double ctl_t::stat_lat Latitude of station [deg].

Definition at line 417 of file libtrac.h.

```
4.2.2.82 double ctl_t::stat_r
Search radius around station [km].
Definition at line 420 of file libtrac.h.
The documentation for this struct was generated from the following file:
    • libtrac.h
4.3
    met_t Struct Reference
Meteorological data.
#include <libtrac.h>
Data Fields
    · double time
          Time [s].
    • int nx
          Number of longitudes.

 int ny

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

    double p [EP]

          Pressure [hPa].

    double ps [EX][EY]

          Surface pressure [hPa].
    float pl [EX][EY][EP]
          Pressure on model levels [hPa].
    float t [EX][EY][EP]
          Temperature [K].
    • float u [EX][EY][EP]
          Zonal wind [m/s].

    float v [EX][EY][EP]

          Meridional wind [m/s].
    float w [EX][EY][EP]
          Vertical wind [hPa/s].
    float h2o [EX][EY][EP]
          Water vapor volume mixing ratio [1].
    float o3 [EX][EY][EP]
          Ozone volume mixing ratio [1].
```

4.3.1 Detailed Description Meteorological data. Definition at line 457 of file libtrac.h. 4.3.2 Field Documentation 4.3.2.1 double met_t::time Time [s]. Definition at line 460 of file libtrac.h. 4.3.2.2 int met_t::nx Number of longitudes. Definition at line 463 of file libtrac.h. 4.3.2.3 int met_t::ny Number of latitudes. Definition at line 466 of file libtrac.h. 4.3.2.4 int met_t::np Number of pressure levels. Definition at line 469 of file libtrac.h. 4.3.2.5 double met_t::lon[EX] Longitude [deg]. Definition at line 472 of file libtrac.h. 4.3.2.6 double met_t::lat[EY] Latitude [deg]. Definition at line 475 of file libtrac.h. 4.3.2.7 double met_t::p[EP] Pressure [hPa]. Definition at line 478 of file libtrac.h.

```
4.3.2.8 double met_t::ps[EX][EY]
Surface pressure [hPa].
Definition at line 481 of file libtrac.h.
4.3.2.9 float met_t::pl[EX][EY][EP]
Pressure on model levels [hPa].
Definition at line 484 of file libtrac.h.
4.3.2.10 float met_t::t[EX][EY][EP]
Temperature [K].
Definition at line 487 of file libtrac.h.
4.3.2.11 float met_t::u[EX][EY][EP]
Zonal wind [m/s].
Definition at line 490 of file libtrac.h.
4.3.2.12 float met_t::v[EX][EY][EP]
Meridional wind [m/s].
Definition at line 493 of file libtrac.h.
4.3.2.13 float met_t::w[EX][EY][EP]
Vertical wind [hPa/s].
Definition at line 496 of file libtrac.h.
4.3.2.14 float met_t::h2o[EX][EY][EP]
Water vapor volume mixing ratio [1].
Definition at line 499 of file libtrac.h.
4.3.2.15 float met_t::o3[EX][EY][EP]
Ozone volume mixing ratio [1].
Definition at line 502 of file libtrac.h.
The documentation for this struct was generated from the following file:
```

· libtrac.h

5 File Documentation 21

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[])

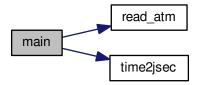
Definition at line 28 of file center.c.

```
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
                 "# $2 = 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)
00068
                "# $10 = altitude (maximum) [km] \n");
00069
       fprintf(out,
    "# $11 = longitude (mean) [deg]\n"
00070
00071
               "# $12 = longitude (sigma) [deg]\n"
00072
                "# $13 = longitude (minimum) [deg]\n"
```

```
"# $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, ".");
                                             /* TODO: Why another "name" here? */
00129
00130
00131
           time2jsec(atoi(year), atoi(mon), atoi(day), atoi(hour), atoi(min), 0, 0,
00132
                        &t);
00133
           00134
00135
00136
00137
                     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
                     atm->lon[atm->np / 4], atm->lon[atm->np / 2],
00143
                     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
00149
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 }
```

5.2 center.c 23

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 = 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"
                  "# $19 = longitude (maximum) [deg]\n");
00079
        fprintf(out,
    "# $20 = latitude (mean) [deg]\n
08000
00081
                  "# $21 = latitude (sigma) [deg]\n"
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
           latm = lats = 0;
00100
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
           /* Get date from filename... */
00122
           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
           day = strtok(NULL, "_");
hour = strtok(NULL, "_");
name = strtok(NULL, "_");
min = strtok(name, ".");
00128
00129
                                            /* TODO: Why another "name" here? */
00130
           time2jsec(atoi(year), atoi(mon), atoi(day), atoi(hour), atoi(min), 0, 0,
00131
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 / 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 / 4], atm->lon[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
```

5.3 dist.c File Reference 25

```
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);
00047
00048
00049
        ALLOC(lon1, double,
00050
                NP);
00051
        ALLOC(lat1, double,
00052
                NP);
        ALLOC(p1, double,
00053
                NP);
00054
00055
        ALLOC(lh1, double,
00056
                NP);
00057
         ALLOC(lv1, double,
00058
                NP);
        ALLOC(lon2, double,
00059
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
          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
        /* Write header... */
00086
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
00108
00109
                  "# $22 = AVTD (maximum) [km] \n"
                  "# $23 = AVTD (maximum trajectory index)\n"  
"# $24 = RVTD (mean) [%%]\n" "# $25 = RVTD (sigma) [%%]\n\n");
00110
00111
00112
00113
        /* Loop over file pairs... */
00114
        for (f = 2; f < argc; f += 2) {
00115
00116
           /* Read atmopheric data... */
          read_atm(argv[f], &ctl, atml);
read_atm(argv[f + 1], &ctl, atm2);
00117
00118
00119
           /* 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;
00130
           rhtd = rhtd2 = 0;
           rvtd = rvtd2 = 0;
00131
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);
geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00136
00137
00138
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
             /* Calculate relative transport deviations... */
             if (f > 2) {
00150
```

```
/* Get trajectory lengths... */
00152
                  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] = atm1->lon[ip];
lat1[ip] = atm1->lat[ip];
00177
00178
00179
               p1[ip] = atm1->p[ip];
00180
00181
               lon2[ip] = atm2 -> lon[ip];
00182
                lat2[ip] = atm2->lat[ip];
00183
               p2[ip] = atm2->p[ip];
00184
00185
             /\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
00188
00189
00190
             /\star Sort distances to calculate percentiles... \star/
00191
             gsl_sort(dh, 1, (size_t) atm1->np);
00192
             gsl_sort(dv, 1, (size_t) atm1->np);
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, ".");
00201
                                                  /* TODO: Why another "name" here? */
00202
00203
             time2jsec(atoi(year), atoi(mon), atoi(day), atoi(hour), atoi(min), 0, 0,
00204
                          &t);
00205
00206
             /* Write output... */
             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 / atml->np - gsl_pow_2(antd / 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,
sqrt(rhtd2 / atml->np - gsl_pow_2(rhtd / atml->np)),
00211
00212
00213
00214
00215
                        avtd / atm1->np,
                       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
00221
00222
00223
          /* Close file... */
00224
          fclose(out);
00225
00226
          /* Free... */
00227
          free(atm1);
00228
          free(atm2);
00229
          free(lon1);
00230
          free(lat1):
00231
          free(p1);
00232
          free(lh1);
00233
          free(lv1);
00234
          free(lon2);
00235
          free(lat2);
00236
          free (p2);
00237
          free(lh2);
```

```
00238 free(lv2);

00239 free(dh);

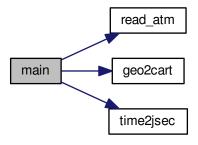
00240 free(dv);

00241

00242 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
          MPTRAC is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by
00004
00005
          the Free Software Foundation, either version 3 of the License, or
00006
00007
          (at your option) any later version.
80000
00009
          \ensuremath{\mathsf{MPTRAC}} is distributed in the hope that it will be useful,
          but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
00012
          GNU General Public License for more details.
00013
00014
          You should have received a copy of the GNU General Public License
00015
          along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>.
00016
00017
          Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026 #include <gsl/gsl_sort.h>
00027
00028 int main(
00029
          int argc,
00030
         char *argv[]) {
00031
00032
          ctl_t ctl;
00033
00034
          atm_t *atm1, *atm2;
00035
00036
          FILE *out;
00037
00038
          char *name, *year, *mon, *day, *hour, *min;
00039
          double aux, x0[3], x1[3], x2[3], \starlon1, \starlat1, \starp1, \starlh1, \starlv1, \starlon2, \starlat2, \starp2, \starlh2, \starlv2, ahtd, avtd, ahtd2, avtd2, rhtd, rvtd, rhtd2, rvtd2, t, \stardh, \stardv;
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);
```

5.4 dist.c 29

```
ALLOC(lat1, double,
00052
               NP);
00053
        ALLOC(p1, double,
00054
               NP);
00055
        ALLOC(lh1, double,
00056
               NP);
        ALLOC(lv1, double,
00058
               NP);
00059
        ALLOC(lon2, double,
00060
              NP);
        ALLOC(lat2, double,
00061
              NP);
00062
00063
        ALLOC(p2, double,
00064
               NP);
00065
        ALLOC(1h2, double,
00066
              NP);
        ALLOC(lv2, 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... */
00083
        if (!(out = fopen(argv[1], "w")))
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"
00104
                 "# $17 = AVTD (10%% percentile) [km]\n"
                 "# $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... */
        for (f = 2; f < argc; f += 2) {
00114
00115
00116
          /* Read atmopheric data... */
00117
          read_atm(argv[f], &ctl, atm1);
read_atm(argv[f + 1], &ctl, atm2);
00118
00119
          /* 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
           /* Init... */
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
```

```
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
00143
                ahtd2 += gsl_pow_2(dh[ip]);
00144
00145
                dv[ip] = fabs(Z(atm1->p[ip]) - Z(atm2->p[ip]));
00146
                avtd += dv[ip];
00147
                avtd2 += gsl_pow_2(dv[ip]);
00148
00149
                /* Calculate relative transport deviations... */
00150
                if (f > 2) {
00151
00152
                   /\star Get trajectory lengths...
                   geo2cart(0, lon1[ip], lat1[ip], x0);
lh1[ip] += DIST(x0, x1);
00153
00154
                   lv1[ip] += fabs(Z(p1[ip]) - Z(atml->p[ip]));
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) atml->np);
gsl_sort(dv, 1, (size_t) atml->np);
00191
00192
00193
00194
              /∗ Get date from filename... ∗/
             for (i = (int) strlen(argv[f]) - 1; argv[f][i] != '/' || i == 0; i--);
name = strtok(&(argv[f][i]), "_");
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
00206
              /* Write output... */
             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

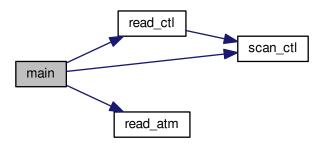
5.5.2.1 int main (int argc, char * argv[])

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> <atm1> [<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
       /* Write info... */
printf("Write trajectory data: %s\n", argv[2]);
00051
00052
00053
00054
        /* Create output file... */
```

```
00055
        if (!(out = fopen(argv[2], "w")))
00056
          ERRMSG("Cannot create file!");
00057
00058
        /* Write header... */
00059
        00060
00061
                "# $2 = altitude [km] \n"
                "# \$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
          fprintf(out, "\n");
00085
00086
00087
88000
        /* Close file... */
00089
        fclose(out);
00090
00091
        /* Free... */
        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.
80000
00009
       MPTRAC is distributed in the hope that it will be useful,
00010
       but WITHOUT ANY WARRANTY; without even the implied warranty of
```

5.7 init.c File Reference 33

```
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
        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 #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")))
00056
          ERRMSG("Cannot create file!");
00057
00058
        /* Write header... */
00059
        fprintf(out,
00060
                "# $1 = time [s] \n"
                "# $2 = altitude [km] \n"
00061
00062
                "# $3 = longitude [deg]\n" "# $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
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
            fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00083
00084
00085
          fprintf(out, "\n");
00086 }
00087
00088
        /* Close file... */
00089
        fclose(out);
00090
00091
        /* Free... */
00092
        free(atm);
00093
00094
        return EXIT_SUCCESS;
00095 }
```

5.7 init.c File Reference

Create atmospheric data file with initial air parcel positions.

Functions

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

5.7.1 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
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
00045
                 /* Check arguments... */
00046
                 if (argc < 3)
                     ERRMSG("Give parameters: <ctl> <atm_out>");
00047
00048
00049
                 /* Read control parameters... */
                read_ctl(argv[1], argc, argv, &ctl);

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

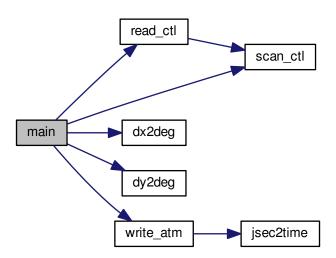
t1 = scan_ctl(argv[1], argc, argv, "INIT_TI", -1, "0", NULL);

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_Z0", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "INIT_Z1", -1, "0", NULL);
dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);
lon0 = scan_ctl(argv[1], argc, argv, "INIT_LONO", -1, "0", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "INIT_LONO", -1, "0", NULL);
dlon = scan_ctl(argv[1], argc, argv, "INIT_LONO", -1, "1", NULL);
00054
00055
00056
00057
00058
00059
00060
                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_SLAT1", -1, "0", NULL);
sx = scan_ctl(argv[1], argc, argv, "INIT_SXT, -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_UT1", -1, "0", NULL);
ulon = scan_ctl(argv[1], argc, argv, "INIT_UT2", -1, "0", NULL);
ulon = scan_ctl(argv[1], argc, argv, "INIT_ULON", -1, "0", NULL);
ulat = scan_ctl(argv[1], argc, argv, "INIT_ULAT1", -1, "0", NULL);
rep = (int) scan_ctl(argv[1], argc, argv, "INIT_MASS1", -1, "0", NULL);
                 lat0 = scan_ctl(argv[1], argc, argv, "INIT_LATO", -1, "0", NULL);
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>
```

5.8 init.c 35

```
/* Set position... */
00087
                      atm->time[atm->np]
00088
                        = (t + gsl_ran_gaussian_ziggurat(rng, st / 2.3548)
00089
                            + ut * (gsl_rng_uniform(rng) - 0.5));
00090
                      atm->p[atm->np]
00091
                        = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
00092
                             + uz * (gsl_rng_uniform(rng) - 0.5));
00093
                      atm->lon[atm->np]
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)
00108
           ERRMSG("Did not create any air parcels!");
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
00120
         gsl_rng_free(rng);
00121
         free (atm);
00122
00123
         return EXIT_SUCCESS;
00124 }
```

Here is the call graph for this function:



5.8 init.c

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

```
00003
00004
             MPTRAC is free software: you can redistribute it and/or modify
00005
             it under the terms of the GNU General Public License as published by
             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
             /\star Check arguments... \star/
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_DLAT1", -1, "1", NULL);
st = scan_ctl(argv[1], argc, argv, "INIT_ST", -1, "0", NULL);
sz = scan_ctl(argv[1], argc, argv, "INIT_SZ,", -1, "0", NULL);
slon = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
00053
00054
00055
00056
00057
00059
00060
00061
00062
00063
            sz = scan_ctl(argv[1], argc, argv, "INIT_SZ", -1, "0", NULL);
slon = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
slat = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
sx = scan_ctl(argv[1], argc, argv, "INIT_SX", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_UT", -1, "0", NULL);
uz = scan_ctl(argv[1], argc, argv, "INIT_UZ", -1, "0", NULL);
ulon = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
ulat = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
rep = (int) scan_ctl(argv[1], argc, argv, "INIT_REP", -1, "1", NULL);
m = scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
00065
00066
00067
00068
00069
00070
00071
00072
00073
00074
00075
             /* Initialize random number generator... */
00076
            asl rna env setup();
00077
             rng = gsl_rng_alloc(gsl_rng_default);
00078
00079
             /* Create grid... */
00080
             for (t = t0; t <= t1; t += dt)</pre>
00081
                for (z = z0; z \le z1; z += dz)
00082
                    for (lon = lon0; lon <= lon1; lon += dlon)</pre>
                        for (lat = lat0; lat <= lat1; lat += dlat)</pre>
00083
00084
                           for (irep = 0; irep < rep; irep++) {</pre>
00085
00086
                               /* Set position... */
00087
                              atm->time[atm->npl
                                 = (t + gsl_ran_gaussian_ziggurat(rng, st / 2.3548)
00088
00089
                                        + ut * (gsl_rng_uniform(rng) - 0.5));
00090
                               atm->p[atm->np]
00091
                                = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
00092
                                         + uz * (gsl_rng_uniform(rng) - 0.5));
00093
                               atm->lon[atm->np]
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)
00104
                       ERRMSG("Too many particles!");
                  }
00105
00106
         /* Check number of air parcels... */
00107
00108
        if (atm->np <= 0)</pre>
00109
           ERRMSG("Did not create any air parcels!");
00110
00111
         /* Initialize mass... */
        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
00116
         /* Save data... */
         write_atm(argv[2], &ctl, atm, t0);
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
         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 %d %g\n", year, mon, day, hour, min, sec, remain);
00045
00046
       return EXIT_SUCCESS;
00047 }
```

Here is the call graph for this function:



5.10 jsec2time.c

```
00001 /*
00002
         This file is part of MPTRAC.
00003
00004
         MPTRAC is free software: you can redistribute it and/or modify
         it under the terms of the GNU General Public License as published by
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
         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
        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
         jsec2time(jsec, &year, &mon, &day, &hour, &min, &sec, &remain);
printf("%d %d %d %d %d %d %g\n", year, mon, day, hour, min, sec, remain);
00043
00044
00045
         return EXIT_SUCCESS;
00046
00047 }
```

5.11 libtrac.c File Reference

MPTRAC library definitions.

Functions

- $\bullet \ \ \text{void cart2geo (double } *x, \ \text{double } *z, \ \text{double } *lon, \ \text{double } *lat)\\$
 - 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.

5.11.2.6 double dy2deg (double dy)

Convert horizontal distance to degrees.

Definition at line 84 of file libtrac.c.

5.11.2.7 double dz2dp (double dz, double p)

Convert vertical distance to pressure.

Definition at line 92 of file libtrac.c.

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

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

Convert geolocation to Cartesian coordinates.

Definition at line 101 of file libtrac.c.

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

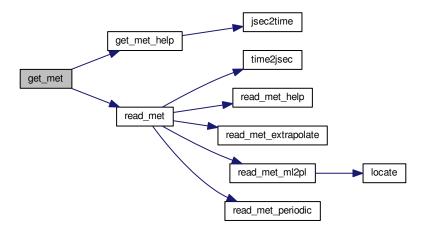
```
5.11.2.9 void get_met ( ctl_t * ctl, char * metbase, double t, met_t * met0, met_t * met1 )
```

Get meteorological data for given timestep.

Definition at line 117 of file libtrac.c.

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

Here is the call graph for this function:



5.11.2.10 void get_met_help (double t, int direct, char * metbase, double dt_met, char * filename)

Get meteorological data for timestep.

Definition at line 156 of file libtrac.c.

```
00161
00162
00163
        double t6, r;
00164
00165
        int year, mon, day, hour, min, sec;
00166
        /* Round time to fixed intervals... */
if (direct == -1)
00167
00168
00169
          t6 = floor(t / dt_met) * dt_met;
00170
00171
          t6 = ceil(t / dt_met) * dt_met;
00172
00173
        /* Decode time... */
00174
        jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00175
        /* Set filename... */
sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", metbase, year, mon, day, hour);
00176
00177
00178 }
```

Here is the call graph for this function:



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

Linear interpolation of 2-D meteorological data.

Definition at line 182 of file libtrac.c.

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

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

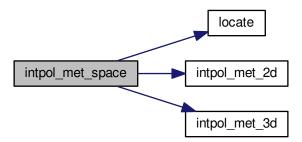
Spatial interpolation of meteorological data.

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);
00260
        iy = locate(met->lat, met->ny, lat);
```

```
00261
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)

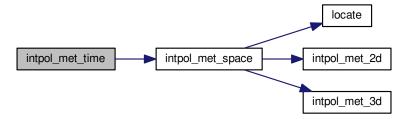
Temporal interpolation of meteorological data.

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
                      t == NULL ? NULL : &t1,
00313
00314
                      u == NULL ? NULL : &u1,
                      v == NULL ? NULL : &v1,
00315
00316
                      w == NULL ? NULL : &w1,
00317
                     h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00318
```

```
00319
        /* 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
       t0.tm_year = 100;
00355
        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
00362
        jsec0 = (time_t) jsec + timegm(&t0);
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
00387
        if (xx[i] < xx[i + 1])
         while (ihi > ilo + 1) {
  i = (ihi + ilo) >> 1;
00388
00389
00390
             if (xx[i] > x)
00391
              ihi = i;
00392
            else
00393
              ilo = i;
00394
        } else
00395
          while (ihi > ilo + 1) {
00396
           i = (ihi + ilo) >> 1;
            if (xx[i] \le x)
00397
00398
              ihi = i;
            else
00399
00400
               ilo = i;
00401
         }
00402
00403
        return ilo;
00404 }
```

5.11.2.17 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... */
00426
          if (!(in = fopen(filename, "r")))
            ERRMSG("Cannot open file!");
00428
00429
          /* Read line... */
00430
          while (fgets(line, LEN, in)) {
00431
            /* Read data... */
TOK(line, tok, "%lg", atm->time[atm->np]);
TOK(NULL, tok, "%lg", atm->p[atm->np]);
TOK(NULL, tok, "%lg", atm->lon[atm->np]);
TOK(NULL, tok, "%lg", atm->lat[atm->np]);
for (iq = 0; iq < ctl->nq; iq++)
    TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00432
00433
00434
00435
00436
00437
00438
00439
00440
             /* Convert altitude to pressure... */
00441
             atm->p[atm->np] = P(atm->p[atm->np]);
00442
            /* Increment data point counter... */
if ((++atm->np) > NP)
00443
00444
               ERRMSG("Too many data points!");
00445
00446
00447
00448
          /* Close file... */
00449
          fclose(in);
00450
00451
          /\star Check number of points... \star/
00452
          if (atm->np < 1)
00453
             ERRMSG("Can not read any data!");
00454 }
```

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

Read control parameters.

Definition at line 458 of file libtrac.c.

```
00462
00463
00464
        int ip, iq;
00465
00466
        /* Write info... */
        printf("\nMassive-Parallel\ Trajectory\ Calculations\ (MPTRAC)\n"
00467
                 "(executable: %s | compiled: %s, %s)\n\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_p = -1;
00476
         ctl->qnt_t = -1;
00477
         ctl->qnt_u = -1;
00478
        ctl \rightarrow qnt_v = -1;
00479
00480
        ctl->qnt w = -1;
00481
         ctl->qnt_h2o = -1;
00482
         ct1->qnt_o3 = -1;
00483
         ctl->qnt\_theta = -1;
00484
         ctl->qnt\_pv = -1;
00485
        ctl->qnt\_tice = -1;
        ctl \rightarrow qnt_tnat = -1;
00486
        ctl->qnt_stat = -1;
00487
00488
00489
         /* Read quantities... */
00490
         ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
         for (iq = 0; iq < ctl->nq; iq++) {
00491
00492
00493
           /* 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",
00494
00495
00496
                     ctl->qnt_format[iq]);
00497
           /* Try to identify quantity... */
if (strcmp(ctl->qnt_name[iq], "m") == 0) {
  ctl->qnt_m = iq;
00498
00499
00500
00501
             sprintf(ctl->qnt_unit[iq], "kg");
           } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
  ctl->qnt_r = iq;
00502
00503
             sprintf(ctl->qnt_unit[iq], "m");
00504
           } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
ctl->qnt_rho = iq;
sprintf(ctl->qnt_unit[iq], "kg/m^3");
00505
00506
00507
00508
           } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
00509
             ctl->qnt_ps = iq;
00510
             sprintf(ctl->qnt_unit[iq], "hPa");
           } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
00511
             ctl->qnt_p = iq;
00513
             sprintf(ctl->qnt_unit[iq], "hPa");
00514
           } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
00515
              ctl->qnt_t = iq;
00516
             sprintf(ctl->qnt_unit[iq], "K");
           } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
  ctl->qnt_u = iq;
00517
00518
             sprintf(ctl->qnt_unit[iq], "m/s");
00520
           } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
             ctl->qnt_v = iq;
00521
             sprintf(ctl->qnt_unit[iq], "m/s");
00522
           } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
  ctl->qnt_w = iq;
00523
00524
             sprintf(ctl->qnt_unit[iq], "hPa/s");
00525
           } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
ctl->qnt_h2o = iq;
00526
00527
             sprintf(ctl->qnt_unit[iq], "1");
00528
           } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
  ctl->qnt_o3 = iq;
  sprintf(ctl->qnt_unit[iq], "1");
00529
00530
00532
           } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
             ctl->qnt_theta = iq;
sprintf(ctl->qnt_unit[iq], "K");
00533
00534
           } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
  ctl->qnt_pv = iq;
00535
00536
00537
             sprintf(ctl->qnt_unit[iq], "PVU");
           } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
```

```
ctl->qnt_tice = iq;
00540
              sprintf(ctl->qnt_unit[iq], "K");
00541
            } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
             ctl->qnt_tnat = iq;
00542
              sprintf(ctl->qnt_unit[iq], "K");
00543
           } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
ctl->qnt_stat = iq;
00544
00546
              sprintf(ctl->qnt_unit[iq], "-");
00547
              scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00548
00549
00550
00551
         /* Time steps of simulation... */
00552
00553
           (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
         if (ctl->direction != -1 && ctl->direction != 1)
   ERRMSG("Set DIRECTION to -1 or 1!");
00554
00555
00556
         ctl->t start =
           scan_ctl(filename, argc, argv, "T_START", -1, "-1e100", 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);
00558
00559
00560
00561
         /* Meteorological data... */
         ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
00562
00563
         ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
         if (ctl->met_np > EP)
00564
00565
           ERRMSG("Too many levels!");
00566
         for (ip = 0; ip < ctl->met_np; ip++)
           ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
00567
00568
00569
         /* Isosurface parameters... */
00570
         ctl->isosurf
         = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00571
00572
00573
00574
         /* Diffusion parameters... */
00575
         ctl->turb dx trop
00576
            = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00577
         ctl->turb_dx_strat
00578
           = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00579
         ctl->turb_dz_trop
           = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00580
00581
         ctl->turb dz strat
00582
           = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00583
         ctl->turb meso =
00584
           scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00585
00586
         /\star Life time of particles... \star/
         ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
00587
         ctl->tdec_strat =
00588
           scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00590
00591
         /* Output of atmospheric data... */
00592
        scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
      atm_basename);
00593
         scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00594
         ctl->atm_dt_out =
00595
           scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00596
         ctl->atm_filter =
00597
            (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
00598
         /* Output of CSI data... */
00599
00600
         scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
00601
        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",
00602
00603
00604
                    ctl->csi_obsfile);
00605
        ctl->csi obsmin =
           scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00607
         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);
00608
00609
00610
00611
         ctl->csi_lon0 =
00612
00613
           scan_ctl(filename, argc, argv, "CSI_LONO", -1, "-180", NULL);
00614
         ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
         ctl->csi_nx =
00615
         (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);
00616
                                                                                      -90", NULL);
00617
00618
00619
         ctl->csi_ny =
00620
            (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
00621
00622
         /* Output of grid data... */
         scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
00623
```

```
ctl->grid_basename);
00624
         scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
00625
       grid_gpfile);
00626 ctl->grid_dt_out =
           scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
00627
00628
         ctl->grid sparse =
         (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
00630
00631
         ctl->grid_nz =
00632
           (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00633
00634
         ctl->grid lon0 =
00635
           scan_ctl(filename, argc, argv, "GRID_LONO", -1, "-180", NULL);
00636
         ctl->grid_lon1
00637
           scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
00638
         ctl->grid_nx =
           (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00639
00640
         ctl->grid lat0 =
00641
           scan_ctl(filename, argc, argv, "GRID_LATO", -1, "-90", NULL);
00642
         ctl->grid_lat1
            scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
00643
00644
         ctl->grid_ny =
           (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
00645
00646
00647
         /* Output of profile data... */
        scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
00648
00649
                    ctl->prof_basename);
00650
         scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
prof_obsfile);

00651 ctil=>=
         ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
00652
00653
         ctl->prof_nz =
00654
            (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
00655
         ctl->prof_lon0 =
00656
           scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
00657
         ctl->prof_lon1
00658
           scan ctl(filename, argc, argv, "PROF LON1", -1, "180", NULL);
         ctl->prof_nx =
00659
00660
            (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00661
         ctl->prof_lat0 =
           scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
00662
00663
         ctl->prof_lat1 =
           scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
00664
00665
         ctl->prof_ny =
00666
           (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00667
00668
         /* Output of station data... */
         scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
00669
00670
                   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);
00671
00672
00673
00674 }
```

Here is the call graph for this function:



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

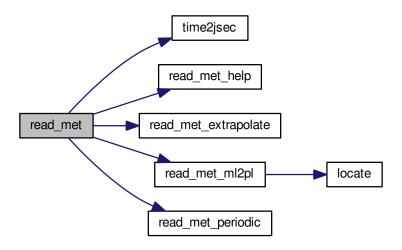
Read meteorological data file.

Definition at line 678 of file libtrac.c.

```
00681
00682
00683
         char tstr[10];
00684
00685
         static float help[EX * EY];
00686
00687
         int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00688
00689
         size_t np, nx, ny;
00690
00691
         /* Write info... */
         printf("Read meteorological data: %s\n", filename);
00692
00693
00694
          /* Get time from filename... */
00695
          sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
         year = atoi(tstr);
sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00696
00697
00698
         mon = atoi(tstr);
         sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00699
00700
         day = atoi(tstr);
00701
          sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00702
         hour = atoi(tstr);
00703
         time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00704
00705
           * Open netCDF file... */
00706
         NC(nc_open(filename, NC_NOWRITE, &ncid));
00707
         /* Get dimensions... */
NC(nc_inq_dimid(ncid, "lon", &dimid));
          /* Get dimensions...
00708
00709
00710
         NC(nc_inq_dimlen(ncid, dimid, &nx));
00711
         if (nx > EX)
00712
            ERRMSG("Too many longitudes!");
00713
00714
         NC(nc_inq_dimid(ncid, "lat", &dimid));
00715
         NC(nc_inq_dimlen(ncid, dimid, &ny));
00716
             (nv > EY)
00717
            ERRMSG("Too many latitudes!");
00718
00719
         NC(nc_inq_dimid(ncid, "lev", &dimid));
00720
         NC(nc_inq_dimlen(ncid, dimid, &np));
00721
          if (np > EP)
00722
           ERRMSG("Too many levels!");
00723
00724
         /* Store dimensions... */
00725
         met->np = (int) np;
00726
         met->nx = (int) nx;
00727
         met->ny = (int) ny;
00728
00729
          /* Get horizontal grid... */
         NC(nc_inq_varid(ncid, "lon", &varid));
00730
         NC(nc_get_var_double(ncid, varid, met->lon));
NC(nc_inq_varid(ncid, "lat", &varid));
00732
00733
         NC(nc_get_var_double(ncid, varid, met->lat));
00734
         /* 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);
00735
00736
00737
00738
00739
00740
00741
00742
00743
          /* Meteo data on pressure levels... */
00744
         if (ctl->met_np <= 0) {</pre>
00745
            /* Read pressure levels from file... */
NC(nc_inq_varid(ncid, "lev", &varid));
00746
00747
            NC(nc_get_var_double(ncid, varid, met->p));
for (ip = 0; ip < met->np; ip++)
  met->p[ip] /= 100.;
00748
00749
00750
00751
00752
            /\star Extrapolate data for lower boundary... \star/
00753
            read_met_extrapolate(met);
00754
00755
00756
          /* Meteo data on model levels... */
00757
00758
            /* Read pressure data from file... */
read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00759
00760
00761
00762
            /\star Interpolate from model levels to pressure levels... \star/
00763
            read_met_ml2pl(ctl, met, met->t);
00764
            read_met_ml2pl(ctl, met, met->u);
00765
            read_met_ml2pl(ctl, met, met->v);
00766
            read_met_ml2pl(ctl, met, met->w);
00767
            read_met_ml2pl(ctl, met, met->h2o);
```

```
read_met_ml2pl(ctl, met, met->o3);
00769
00770
              /\star Set pressure levels... \star/
             met->np = ctl->met_np;
for (ip = 0; ip < met->np; ip++)
  met->p[ip] = ctl->met_p[ip];
00771
00772
00773
00774
00775
00776
           /\star Check ordering of pressure levels... \star/
           for (ip = 1; ip < met->np; ip++)
  if (met->p[ip - 1] < met->p[ip])
    ERRMSG("Pressure levels must be descending!");
00777
00778
00779
00780
           /* Read surface pressure... */
if (nc_ing_varid(ncid, "PS", &varid) == NC_NOERR) {
00781
00782
             NC(nc_get_var_float(ncid, varid, help));
00783
          for (iy = 0; iy < met->ny; iy++)
    for (ix = 0; ix < met->nx; ix++)
        met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
} else if (nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00784
00785
00786
00787
00788
             NC(nc_get_var_float(ncid, varid, help));
             for (iy = 0; iy < met->ny; iy++)
  for (ix = 0; ix < met->nx; ix++)
00789
00790
00791
                  met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00792
           } else
00793
             for (ix = 0; ix < met->nx; ix++)
00794
                for (iy = 0; iy < met->ny; iy++)
00795
                   met->ps[ix][iy] = met->p[0];
00796
00797
          /\star Create periodic boundary conditions... \star/
00798
          read_met_periodic(met);
00799
00800
           /* Close file... */
00801
           NC(nc_close(ncid));
00802 }
```

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 806 of file libtrac.c.

```
00807
00808
00809
         int ip, ip0, ix, iy;
00810
00811
         /* Loop over columns... */
00812
         for (ix = 0; ix < met->nx; ix++)
           for (iy = 0; iy < met->ny; iy++) {
00814
00815
               /\star Find lowest valid data point... \star/
              for (ip0 = met->np - 1; ip0 >= 0; ip0--)
   if (!gsl_finite(met->t[ix][iy][ip0])
00816
00817
00818
                     | | !gsl_finite(met->u[ix][iy][ip0])
                     || !gsl_finite(met->v[ix][iy][ip0])
00819
00820
                      || !gsl_finite(met->w[ix][iy][ip0]))
00821
                   break;
00822
00823
              /* Extrapolate... */
             /* Extrapolate... */
for (ip = ip0; ip >= 0; ip--) {
    met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
00824
00825
                met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
00827
                met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
00828
                met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
                met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00829
00830
00831
              }
00833 }
```

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 837 of file libtrac.c.

```
00844
00845
        static float help[EX * EY * EP];
00846
00847
        int ip, ix, iv, n = 0, varid;
00848
00849
         /* Check if variable exists... */
00850
        if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00851
         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00852
             return:
00853
         /* Read data...
00854
00855
        NC(nc_get_var_float(ncid, varid, help));
00856
00857
         /* Copy and check data... */
        for (ip = 0; ip < met->np; ip++)
  for (iy = 0; iy < met->ny; iy++)
00858
00859
             for (ix = 0; ix < met->nx; ix++) {
  dest[ix][iy][ip] = scl * help[n++];
00860
00862
               if (fabs(dest[ix][iy][ip] / scl) > 1e14)
00863
                  dest[ix][iy][ip] = GSL_NAN;
00864
             }
00865 }
```

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 869 of file libtrac.c.

```
00872 {
00873
00874 double aux[EP], p[EP], pt;
00875
00876 int ip, ip2, ix, iy;
00877
00878 /* Loop over columns... */
00879 for (ix = 0; ix < met->nx; ix++)
00880 for (iy = 0; iy < met->ny; iy++) {
```

```
/* Copy pressure profile... */
              for (ip = 0; ip < met->np; ip++)
  p[ip] = met->pl[ix][iy][ip];
00883
00884
00885
              /* Interpolate... */
for (ip = 0; ip < ctl->met_np; ip++) {
   pt = ctl->met_p[ip];
00886
00887
00888
00889
                if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
00890
                  pt = p[0];
                else if ((pt > p[met->np - 1] && p[1] > p[0])
  || (pt < p[met->np - 1] && p[1] < p[0]))
00891
00892
                  pt = p[met->np - 1];
00893
                00894
00895
00896
00897
00898
00899
              /* Copy data... */
for (ip = 0; ip < ctl->met_np; ip++)
00900
00901
                var[ix][iy][ip] = (float) aux[ip];
00902
00903 }
```

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 907 of file libtrac.c.

```
00908
00909
00910
          int ip, iy;
00912
          /* Check longitudes... */
00913
          if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
00914
                          + met - > lon[1] - met - > lon[0] - 360) < 0.01))
00915
             return:
00916
00917
          /* Increase longitude counter... */
00918
          if ((++met->nx) > EX)
00919
             ERRMSG("Cannot create periodic boundary conditions!");
00920
         /* Set longitude... */
met->lon[met->nx - 2] + met->lon[1] - met->
00921
00922
        lon[0];
00923
00924
           /* Loop over latitudes and pressure levels... */
00925
           for (iy = 0; iy < met->ny; iy++)
             for (ip = 0; ip < met >ny, 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];
00926
00927
00928
00929
                met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
00930
00931
               met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
00932
00933
00934
00935 }
```

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 939 of file libtrac.c.

```
00946
00947
00948
        FILE *in = NULL;
00949
00950
        char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
00951
          msg[LEN], rvarname[LEN], rval[LEN];
00952
00953
        int contain = 0, i;
00954
00955
         /* Open file... */
        if (filename[strlen(filename) - 1] != '-')
if (!(in = fopen(filename, "r")))
00956
00957
             ERRMSG("Cannot open file!");
00959
00960
         /* Set full variable name... */
         if (arridx >= 0) {
00961
         sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
00962
00963
00964
        sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
00965
00966
00967
00968
00969
        /* Read data... */
        if (in != NULL)
00970
         while (fgets(line, LEN, in))
00971
00972
            if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
00973
               if (strcasecmp(rvarname, fullname1) == 0 | |
00974
                    strcasecmp(rvarname, fullname2) == 0) {
00975
                 contain = 1;
00976
                 break;
00977
               }
00978
        for (i = 1; i < argc - 1; i++)</pre>
        if (strcasecmp(argv[i], fullname1) == 0 ||
    strcasecmp(argv[i], fullname2) == 0) {
    sprintf(rval, "%s", argv[i + 1]);
00979
00980
00981
00982
            contain = 1:
00983
             break;
00984
00985
00986
        /* Close file... */
if (in != NULL)
00987
00988
          fclose(in);
00990
        /* Check for missing variables... */
00991
        if (!contain) {
         if (strlen(defvalue) > 0)
00992
00993
             sprintf(rval, "%s", defvalue);
00994
          else {
00995
            sprintf(msg, "Missing variable %s!\n", fullname1);
00996
             ERRMSG(msg);
00997
00998
00999
        /* Write info... */
01000
        printf("%s = %s\n", fullname1, rval);
01001
01003
         /* Return values... */
01004
        if (value != NULL)
01005
          sprintf(value, "%s", rval);
01006
        return atof(rval);
01007 }
```

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 1011 of file libtrac.c.

```
01020
01021
       struct tm t0, t1;
01022
       t0.tm_year = 100;
01023
01024
       t0.tm mon = 0;
       t0.tm_mday = 1;
01025
01026
        t0.tm\_hour = 0;
01027
       t0.tm_min = 0;
01028
       t0.tm_sec = 0;
01029
       t1.tm_year = year - 1900;
01030
       t1.tm_mon = mon - 1;
01031
01032
       t1.tm_mday = day;
01033
       t1.tm_hour = hour;
01034
       t1.tm_min = min;
       t1.tm_sec = sec;
01035
01036
       *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01038 }
```

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

Measure wall-clock time.

Definition at line 1042 of file libtrac.c.

```
01045
                   {
01046
01047
        static double starttime[NTIMER], runtime[NTIMER];
01048
       /* Check id... */
if (id < 0 || id >= NTIMER)
01049
01050
01051
          ERRMSG("Too many timers!");
01052
01053
        /* Start timer... */
        if (mode == 1) {
01054
         if (starttime[id] <= 0)</pre>
01055
01056
           starttime[id] = omp_get_wtime();
          else
01058
            ERRMSG("Timer already started!");
01059
01060
        /* Stop timer... */
else if (mode == 2) {
01061
01062
01063
        if (starttime[id] > 0) {
01064
            runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
01065
            starttime[id] = -1;
01066
            ERRMSG("Timer not started!");
01067
01068
01069
01070
       /* Print timer... */
01071
        else if (mode == 3)
          printf("%s = %g s\n", name, runtime[id]);
01072
01073 }
```

5.11.2.27 double tropopause (double t, double lat)

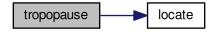
Definition at line 1077 of file libtrac.c.

```
01080
01081
                  static double doys[12]
                  = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01082
01083
                  static double lats[73]
01084
                      static double lats[73]
= { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5, -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5, -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5, -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5, 15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5, 45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5, 75, 77.5, 80, 82.5, 85, 87.5, 90
01085
01086
01087
01088
01089
01090
01091
01092
```

```
01094
             static double tps[12][73]
01095
                = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
01096
                         297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
01097
                         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,
01098
                         152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
01100
01101
                         277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
             275.3, 275.6, 275.4, 274.1, 273.5}, {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7, 300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
01102
01103
01104
               150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
01105
               98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
01106
01107
               98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
01108
               220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
               284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01109
               287.5, 286.2, 285.8},
01110
             {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
01111
               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,
01113
01114
               100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01115
               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,
01116
01117
               304.3, 304.9, 306, 306.6, 306.2, 306},
01118
              {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
01119
01120
               290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
               195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7, 102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79, 99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
01121
01122
01123
01124
               148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6,
               263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.
01125
01126
               315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
             {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8, 260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9, 205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7, 101.8, 101.4, 101.1, 101, 101, 101.1, 101.2, 101.5, 101.9,
01127
01128
01129
01131
               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,
01132
01133
               273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
               325.3, 325.8, 325.8},
01134
             323.3, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 32
01135
01136
01138
01139
               106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
01140
               127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
             251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9, 308.5, 312.2, 313.1, 313.3}, {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
01141
01142
01143
               187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
01144
01145
               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,
01146
01147
               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,
01148
               275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
01150
             {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01151
01152
               185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
01153
               233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
              110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.6, 111.9, 112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
01154
               120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
01156
01157
               230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7
01158
               278.2, 282.6, 287.4, 290.9, 292.5, 293},
01159
             {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
               183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
01160
               243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
01161
               114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4,
               110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
01163
01164
               114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
               203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5, 276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
01165
01166
             {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,
01167
               237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7,
01169
                                                                                                  124.8,
01170
               111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01171
               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,
01172
               206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01173
               279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
               305.1},
01175
              {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01176
01177
               253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
               223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9, 108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
01178
```

```
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,
01182
            286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6}, {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7, 284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192, 175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1, 100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
01183
01184
01185
01186
01187
             100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160, 186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278, 280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4, 281.7, 281.1, 281.2}
01188
01189
01190
01191
01192
01193
01194
            double doy, p0, p1, pt;
01195
01196
            int imon, ilat:
01197
           /* Get day of year... */
01198
            doy = fmod(t / 86400., 365.25);
01199
01200
            while (doy < 0)
01201
               doy += 365.25;
01202
01203
            /* Get indices... */
           imon = locate(doys, 12, doy);
ilat = locate(lats, 73, lat);
01204
01205
01206
01207
             /* Get tropopause pressure... */
           01208
01209
            p1 = LIN(lats[ilat], tps[imon + 1][ilat],
lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
01210
01211
01212
            pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01213
01214
             /* Return tropopause pressure... */
01215
            return pt;
01216 }
```

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 1220 of file libtrac.c.

```
01224
01225
01226
        FILE *in, *out;
01227
01228
        char line[LEN];
01229
01230
        double r, t0, t1;
01231
01232
        int ip, iq, year, mon, day, hour, min, sec;
01233
01234
         /* Set time interval for output... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01235
01236
01237
01238
        /* Check if gnuplot output is requested... */
        if (ctl->atm_gpfile[0] != '-') {
```

```
01240
01241
           /* Write info... */
01242
           printf("Plot atmospheric data: %s.png\n", filename);
01243
01244
          /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
01245
01246
             ERRMSG("Cannot create pipe to gnuplot!");
01247
          /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
01248
01249
01250
01251
           /* Set time string... */
           jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01252
01253
01254
                    year, mon, day, hour, min);
01255
           /* Dump gnuplot file to pipe... */
01256
          if (!(in = fopen(ctl->atm_gpfile, "r")))
ERRMSG("Cannot open file!");
01257
01258
01259
          while (fgets(line, LEN, in))
             fprintf(out, "%s", line);
01260
01261
          fclose(in);
01262
01263
01264
        else {
01265
01266
           /\star Write info... \star/
01267
          printf("Write atmospheric data: %s\n", filename);
01268
01269
           /* Create file... */
          if (!(out = fopen(filename, "w")))
01270
01271
             ERRMSG("Cannot create file!");
01272
01273
01274
        /* Write header... */
        01275
01276
                 "# $2 = altitude [km] \n"
01278
                 "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
        01279
01280
01281
01282
01283
01284
         /* Write data... */
01285
         for (ip = 0; ip < atm->np; ip++) {
01286
          /* Check time... */
if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
01287
01288
01289
             continue:
01290
           /* Write output... */
01291
          fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
    atm->lon[ip], atm->lat[ip]);
01292
01293
           for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01294
01295
01296
01297
01298
           fprintf(out, "\n");
01299
01300
        /* Close file... */
01301
01302
        fclose(out);
01303 }
```

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 1307 of file libtrac.c.

```
01311
                    {
01312
01313
        static FILE *in, *out;
01314
01315
        static char line[LEN]:
01316
        static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01317
01318
          rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01319
01320
        static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01321
01322
         /* Init... */
01323
        if (!init) {
01324
          init = 1;
01325
01326
           /* Check quantity index for mass... */
01327
           if (ctl->qnt_m < 0)
             ERRMSG("Need quantity mass to analyze CSI!");
01328
01329
01330
           /* Open observation data file... */
01331
           printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
01332
           if (!(in = fopen(ctl->csi_obsfile, "r")))
             ERRMSG("Cannot open file!");
01333
01334
01335
           /* Create new file... */
          /* Create new IIIe... ^/
printf("Write CSI data: %s\n", filename);
if (!/out = fonen(filename, "w")))
01336
01337
01338
            ERRMSG("Cannot create file!");
01339
01340
           /* Write header... */
01341
          fprintf(out,
                    "# $1 = time [s]\n"
01342
01343
                   "# $2 = number of hits (cx) \n"
01344
                    "# $3 = number of misses (cy) \n"
01345
                    "# $4 = number of false alarms (cz)\n"
                    "# $5 = number of observations (cx + cy) \n"
01346
                    "# $6 = number of forecasts (cx + cz) \n"
01347
                    "# $7 = bias (forecasts/observations) [%%]\n"
01348
                   "# $8 = probability of detection (POD) [%%]\n"
"# $9 = false alarm rate (FAR) [%%]\n"
01349
01350
01351
                    "# $10 = critical success index (CSI) [%%]\n\n");
01352
01353
01354
        /* Set time interval... */
        t0 = t - 0.5 * ct1->dt_mod;
t1 = t + 0.5 * ct1->dt_mod;
01355
01356
01357
        /\star Initialize grid cells... \star/
01358
        for (ix = 0; ix < ctl->csi_nx; ix++)
  for (iy = 0; iy < ctl->csi_ny; iy++)
    for (iz = 0; iz < ctl->csi_nz; iz++)
01359
01360
01361
01362
               modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01363
01364
         /* Read data... */
01365
        while (fgets(line, LEN, in)) {
01366
01367
           /* Read data... *
           if (sscanf(line, "%lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01368
01369
01370
             continue;
01371
           /* Check time... */
01372
01373
           if (rt < t0)
01374
             continue;
01375
           if (rt > t1)
01376
            break;
01377
01378
           /* Calculate indices... */
01379
          ix = (int) ((rlon - ctl->csi_lon0)
                         / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01380
01381
           iy = (int) ((rlat - ctl -> csi_lat0))
01382
                         / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
           iz = (int) ((rz - ctl -> csi_z0)
01383
01384
                        / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01385
01386
           /* Check indices... */
01387
           if (ix < 0 || ix >= ctl->csi_nx ||
```

```
iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01389
01390
01391
           /* Get mean observation index... */
01392
          obsmean[ix][iy][iz] += robs;
          obscount[ix][iy][iz]++;
01393
01394
01395
01396
        /* Analyze model data... ∗/
01397
        for (ip = 0; ip < atm->np; ip++) {
01398
01399
          /* Check time... */
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
01400
01401
            continue;
01402
          /* Get indices... */
ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01403
01404
                        / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01405
          iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
01406
                        / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01407
          01408
01409
01410
          /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
01411
01412
              iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01413
01414
01415
01416
          /* Get total mass in grid cell... */
          modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01417
01418
01419
01420
        /* Analyze all grid cells... */
01421
        for (ix = 0; ix < ctl->csi_nx; ix++)
         for (iy = 0; iy < ctl->csi_ny; iy++)
  for (iz = 0; iz < ctl->csi_nz; iz++) {
01422
01423
01424
               /\star Calculate mean observation index... \star/
01426
               if (obscount[ix][iy][iz] > 0)
01427
                obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01428
01429
               /* Calculate column density... */
              if (modmean[ix][iy][iz] > 0) {
  dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
  dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
01430
01431
01432
01433
                 lat = ctl->csi\_lat0 + dlat * (iy + 0.5);
                area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180. * cos(lat * M_PI / 180.);
01434
01435
                modmean[ix][iy][iz] /= (1e6 * area);
01436
01437
01438
01439
               /* Calculate CSI... */
01440
               if (obscount[ix][iy][iz] > 0) {
01441
               if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
                     modmean[ix][iy][iz] >= ctl->csi_modmin)
01442
01443
                   cx++;
                else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01445
                          modmean[ix][iy][iz] < ctl->csi_modmin)
01446
                 01447
01448
01449
                   cz++;
01450
              }
01451
01452
01453
        /* Write output... */
01454
        if (fmod(t, ctl->csi_dt_out) == 0) {
01455
          /* Write... */
fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
01456
01457
                  (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN, (cx + cz > 0) ? (100. * cx) / (cx + cz) : GSL_NAN,
01458
01459
01460
01461
                   (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01462
01463
01464
          /* Set counters to zero... */
01465
          cx = cy = cz = 0;
01466
01467
        /* Close file... */
01468
        if (t == ctl->t_stop)
01469
01470
          fclose(out);
01471 }
```

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

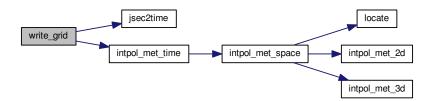
Write gridded data.

Definition at line 1475 of file libtrac.c.

```
01481
                      {
01482
01483
         FILE *in, *out;
01484
01485
         char line[LEN];
01486
01487
         static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
            area, rho_air, press, temp, cd, mmr, t0, t1, r;
01489
01490
         static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01491
         /* Check dimensions... */
if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
    ERRMSG("Grid dimensions too large!");
01492
01493
01494
01495
01496
          /\star Check quantity index for mass... \star/
01497
         if (ctl->qnt_m < 0)
           ERRMSG("Need quantity mass to write grid data!");
01498
01499
01500
          /* Set time interval for output... */
         t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01501
01502
01503
01504
          /* Set grid box size... */
         dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
01505
         dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
01506
01508
01509
          /* Initialize grid... */
         for (ix = 0; ix < ctl->grid_nx; ix++)
  for (iy = 0; iy < ctl->grid_ny; iy++)
    for (iz = 0; iz < ctl->grid_nz; iz++)
01510
01511
01512
01513
                 grid_m[ix][iy][iz] = 0;
01515
          /* Average data... */
         for (ip = 0; ip < atm->np; ip++)
  if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {</pre>
01516
01517
01518
               /* Get index... */
              ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
01520
01521
01522
              iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01523
01524
               /* Check indices... */
              if (ix < 0 || ix >= ctl->grid_nx ||
01525
                   iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01526
01527
01528
              /* Add mass... */
grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01529
01530
01531
01532
01533
          /* Check if gnuplot output is requested... */
01534
         if (ctl->grid_gpfile[0] != '-') {
01535
            /* Write info... */
01536
01537
            printf("Plot grid data: %s.png\n", filename);
01539
             /* Create gnuplot pipe... */
            if (!(out = popen("gnuplot", "w")))
01540
              ERRMSG("Cannot create pipe to gnuplot!");
01541
01542
01543
            /* Set plot filename... */
            fprintf(out, "set out \"%s.png\"\n", filename);
01544
01545
            /* Set time string... */
01546
            jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01547
01548
01549
                      year, mon, day, hour, min);
01550
01551
            /\star Dump gnuplot file to pipe... \star/
            if (!(in = fopen(ctl->grid_gpfile, "r")))
    ERRMSG("Cannot open file!");
01552
01553
            while (fgets(line, LEN, in))
  fprintf(out, "%s", line);
01554
01555
01556
            fclose(in);
01557
```

```
01558
01559
01560
01561
           /* Write info... */
01562
           printf("Write grid data: %s\n", filename);
01563
01564
            /* Create file... */
01565
            if (!(out = fopen(filename, "w")))
              ERRMSG("Cannot create file!");
01566
01567
01568
01569
         /* Write header... */
01570
         fprintf(out,
01571
                   "# $1 = time [s] \n"
                   "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n"
"# $4 = latitude [deg]\n"
01572
01573
01574
                   "# $5 = surface area [km^2]\n"
"# $6 = layer width [km]\n"
01575
01576
01577
                   "# $7 = temperature [K]\n"
01578
                   "# $8 = \text{column density } [kg/m^2] \n"
                   "# $9 = mass mixing ratio [1]\n\n");
01579
01580
         /* Write data... */
for (ix = 0; ix < ctl->grid_nx; ix++) {
   if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01581
01582
01584
              fprintf(out, "\n");
            for (iy = 0; iy < ctl->grid_ny; iy++) {
   if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
01585
01586
              fprintf(out, "\n");
for (iz = 0; iz < ctl->grid_nz; iz++)
01587
01588
01589
                if (!ctl->grid_sparse
01590
                      || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01591
                   /* Set coordinates... */
z = ctl->grid_z0 + dz * (iz + 0.5);
lon = ctl->grid_lon0 + dlon * (ix + 0.5);
01592
01593
01594
                   lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01595
01596
01597
                   /* Get pressure and temperature... */
01598
                   press = P(z);
                   01599
01600
01601
01602
                   /* Calculate surface area... */
                   area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
 * cos(lat * M_PI / 180.);
01603
01604
01605
01606
                   /* Calculate column density... */
                   cd = grid_m[ix][iy][iz] / (1e6 * area);
01607
01608
                  /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01609
01610
01611
01612
01613
                    /* Write output...
                   fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
01614
01615
                             t, z, lon, lat, area, dz, temp, cd, mmr);
01616
01617
           }
01618
         }
01619
01620
          /* Close file... */
01621
         fclose(out);
01622 }
```

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)

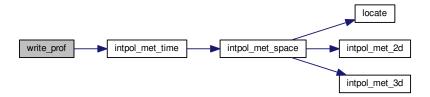
Write profile data.

Definition at line 1626 of file libtrac.c.

```
01632
                     {
01633
01634
         static FILE *in, *out;
01635
01636
        static char line[LEN]:
01637
        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,
01638
01639
01640
          press, temp, rho_air, mmr, h2o, o3;
01641
01642
        static int init, obscount[GX][GY], ip, ix, iy, iz;
01643
01644
         /* Init... */
01645
         if (!init) {
01646
           init = 1;
01647
01648
           /\star Check quantity index for mass... \star/
           if (ctl->qnt_m < 0)</pre>
01649
             ERRMSG("Need quantity mass!");
01650
01651
01652
            /* Check dimensions... */
01653
           01654
             ERRMSG("Grid dimensions too large!");
01655
01656
           /* Open observation data file... */
           printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01657
               (!(in = fopen(ctl->prof_obsfile, "r")))
01659
             ERRMSG("Cannot open file!");
01660
           /* Create new file... */
printf("Write profile data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01661
01662
01663
             ERRMSG("Cannot create file!");
01664
01665
01666
            /* Write header... */
01667
           fprintf(out,
                     "# $1
                            = time [s]\n"
01668
                     "# $2
                            = altitude [km]\n"
01669
                    "# $3
                            = longitude [deg]\n"
01670
01671
                    "# $4 = latitude [deg]\n"
01672
                     "# $5
                            = pressure [hPa]\n"
01673
                     "# $6 = temperature [K]\n"
                     "# $7
01674
                            = mass mixing ratio [1]\n"
                    "# $8 = H2O volume mixing ratio [1]\n"
"# $9 = 03 volume mixing ratio [1]\n"
01675
01676
01677
                    "# $10 = mean BT index [K]\n");
01678
01679
           /* Set grid box size... */
           dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
01680
01681
01682
01683
01684
01685
         /\star Set time interval... \star/
         t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01686
01687
01688
01689
         /* Initialize... */
01690
         for (ix = 0; ix < ctl->prof_nx; ix++)
01691
           for (iy = 0; iy < ctl->prof_ny; iy++) {
              obsmean[ix][iy] = 0;
01692
              obscount[ix][iy] = 0;
01693
             tmean[ix][iy] = 0;
for (iz = 0; iz < ctl->prof_nz; iz++)
01694
01695
                mass[ix][iy][iz] = 0;
01696
01697
01698
         /* Read data... */
while (fgets(line, LEN, in)) {
01699
01700
01701
01702
01703
           if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01704
             continue;
01705
01706
           /* Check time... */
01707
           if (rt < t0)</pre>
01708
             continue;
```

```
01709
           if (rt > t1)
01710
             break;
01711
           /* Calculate indices... */
ix = (int) ((rlon - ctl->prof_lon0) / dlon);
iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01712
01713
01714
01715
01716
            /* Check indices... */
01717
           if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01718
             continue;
01719
01720
           /* Get mean observation index... */
           obsmean[ix][iy] += robs;
tmean[ix][iy] += rt;
01721
01722
01723
           obscount[ix][iy]++;
01724
01725
01726
         /* Analyze model data... */
         for (ip = 0; ip < atm->np; ip++) {
01728
01729
            /* Check time... */
01730
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
01731
             continue;
01732
01733
           /* Get indices... */
01734
           ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
           iy = (int) ((atm->laf[ip] - ctl->prof_lat0) / dlat);
iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01735
01736
01737
           /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx ||
01738
01739
                iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01741
01742
           /* Get total mass in grid cell... */
mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01743
01744
01745
01746
01747
         /* Extract profiles... */
01748
         for (ix = 0; ix < ctl->prof_nx; ix++)
01749
           for (iy = 0; iy < ctl->prof_ny; iy++)
01750
             if (obscount[ix][iy] > 0) {
01751
01752
                /* Write output... */
01753
                fprintf(out, "\n");
01754
01755
                /* Loop over altitudes... */
01756
                for (iz = 0; iz < ctl->prof_nz; iz++) {
01757
01758
                  /* 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);
01760
01761
01762
01763
                  /* Get meteorological data... */
                   press = P(z);
01764
01765
                   intpol_met_time(met0, met1, t, press, lon, lat,
01766
                                     NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
01767
                  /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
 * cos(lat * M_PI / 180.);
01768
01769
01770
01771
01772
                  mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
01773
                  01774
01775
                            tmean[ix][iy] / obscount[ix][iy],
01776
                            z, lon, lat, press, temp, mmr, h2o, o3, obsmean[ix][iy] / obscount[ix][iy]);
01777
01778
01779
01780
01781
         /* Close file... */
01782
        if (t == ctl->t_stop)
01783
01784
           fclose(out);
01785 }
```

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 1789 of file libtrac.c.

```
01793
                   {
01794
01795
        static FILE *out;
01797
        static double rmax2, t0, t1, x0[3], x1[3];
01798
01799
        static int init, ip, iq;
01800
01801
        /* Init... */
01802
        if (!init) {
01803
          init = 1;
01804
01805
          /* Write info... */
          printf("Write station data: %s\n", filename);
01806
01807
01808
           /* Create new file... */
01809
          if (!(out = fopen(filename, "w")))
01810
            ERRMSG("Cannot create file!");
01811
           /* Write header... */
01812
01813
          fprintf(out,
                   "# $1 = time [s]\n"
"# $2 = altitude [km]\n"
01814
01815
01816
                   "# $3 = longitude [deg] \n" "# $4 = latitude [deg] \n");
           01817
01818
01819
01820
          fprintf(out, "\n");
01821
01822
           /* Set geolocation and search radius... */
01823
           geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
01824
           rmax2 = gsl_pow_2(ctl->stat_r);
01825
01826
01827
         /* Set time interval for output... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01828
01829
01830
        /* Loop over air parcels... */
for (ip = 0; ip < atm->np; ip++) {
01831
01832
01833
01834
01835
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
01836
             continue;
01837
          /* Check station flag... */
if (ctl->qnt_stat >= 0)
01838
01839
01840
            if (atm->q[ctl->qnt_stat][ip])
01841
01842
01843
           /* Get Cartesian coordinates... */
          geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
01844
01845
01846
           /\star Check horizontal distance... \star/
```

5.12 libtrac.c 67

```
01847
          if (DIST2(x0, x1) > rmax2)
01848
            continue;
01849
          /* Set station flag... */
if (ctl->qnt_stat >= 0)
atm->q[ctl->qnt_stat][ip] = 1;
01850
01851
01852
01853
01854
           /* Write data... */
          01855
01856
          for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01857
01858
01859
            fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01860
01861
          fprintf(out, "\n");
01862
01863
        /* Close file... */
if (t == ctl->t_stop)
01864
01865
01866
          fclose(out);
01867 }
```

Here is the call graph for this function:



5.12 libtrac.c

```
00001 /*
00002
       This file is part of MPTRAC.
00003
00004
       \ensuremath{\mathsf{MPTRAC}} is free software: you can redistribute it and/or modify
00005
       it under the terms of the GNU General Public License as published by
00006
       the Free Software Foundation, either version 3 of the License, or
00007
       (at your option) any later version.
00008
00009
       \ensuremath{\mathsf{MPTRAC}} is distributed in the hope that it will be useful,
       but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
00012
       GNU General Public License for more details.
00013
00014
       You should have received a copy of the GNU General Public License
00015
       along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>.
00016
00017
       Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00028
00029 void cart2geo(
00030
       double *x,
00031
       double *z,
00032
       double *lon,
00033
       double *lat) {
00034
00035
       double radius:
00036
00037
       radius = NORM(x);
00038
       *lat = asin(x[2] / radius) * 180 / M_PI;
00039
       *lon = atan2(x[1], x[0]) * 180 / M_PI;
00040
       *z = radius - RE;
00041 }
00042
00044
```

```
00045 double deg2dx(
00046
     double dlon,
00047
      double lat) {
00048
00049
      return dlon * M PI * RE / 180. * cos(lat / 180. * M PI);
00050 }
00053
00054 double deg2dy(
00055
     double dlat) {
00056
00057
     return dlat * M_PI * RE / 180.;
00058 }
00059
00061
00062 double dp2dz(
00063
     double dp,
00064
     double p) {
00065
00066
      return -dp * H0 / p;
00067 }
00068
00070
00071 double dx2deg(
00072 double dx,
00073
     double lat)
00074
     /* Avoid singularity at poles... */
if (lat < -89.999 || lat > 89.999)
00075
00076
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
      return dy * 180. / (M_PI * RE);
00087
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,
00104
     double lat,
00105
     double *x) {
00106
00107
     double radius;
00108
00109
      radius = z + RE;
     x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
x[2] = radius * sin(lat / 180 * M_PI);
00110
00111
00112
00113 }
00114
00116
00117 void get_met(
     ctl_t * ctl,
char *metbase,
00118
00119
00120
     double t,
00121
     met_t * met0,
00122
     met_t * met1) {
00123
00124
     char filename[LEN1:
00125
00126
     static int init;
00127
00128
      /* Init... */
00129
      if (!init) {
00130
       init = 1;
00131
```

5.12 libtrac.c 69

```
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);
    read_met(ctl, filename, met1);
00136
00137
00138
00139
       /* Read new data for forward trajectories... */
00140
       if (t > met1->time && ct1->direction == 1) {
00141
        memcpy(met0, met1, sizeof(met_t));
00142
         get_met_help(t, 1, metbase, ctl->dt_met, filename);
00143
         read_met(ctl, filename, met1);
00144
00145
00146
       /* Read new data for backward trajectories... */
00147
       if (t < met0->time && ctl->direction == -1) {
        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,
00160
       double dt_met,
00161
       char *filename) {
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
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
       sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", metbase, year, mon, day, hour);
00178 }
00179
00181
00182 void intpol_met_2d(
00183
       double array[EX][EY],
00184
       int ix,
00185
       int iy,
       double wx,
00186
00187
       double wy,
00188
       double *var) {
00189
00190
       double aux00, aux01, aux10, aux11;
00191
00192
       /* Set variables...
00193
       aux00 = array[ix][iy];
00194
       aux01 = array[ix][iy + 1];
       aux10 = array[ix + 1][iy];
00195
00196
       aux11 = array[ix + 1][iy + 1];
00197
00198
       /* Interpolate horizontally... */
       aux00 = wy * (aux00 - aux01) + aux01;
aux11 = wy * (aux10 - aux11) + aux11;
00199
00200
00201
       *var = wx * (aux00 - aux11) + aux11;
00202 }
00203
00205
00206 void intpol_met_3d(
00207
       float array[EX][EY][EP],
00208
       int ip,
00209
       int ix,
       int iy,
00210
00211
       double wp,
00212
       double wx,
00213
       double wy,
00214
       double *var) {
00215
       double aux00, aux01, aux10, aux11;
00216
00217
```

```
00218
        /* Interpolate vertically... */
00219
        aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00220
         + array[ix][iy][ip + 1];
        aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
00221
        + array[ix][iy + 1][ip + 1];
aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
+ array[ix + 1][iy][ip + 1];
00222
00223
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, double lon,
00238
00239
00240
        double lat,
00241
        double *ps,
00242
        double *t,
00243
        double *u,
00244
        double *v,
00245
        double *w,
        double *h2o,
00246
00247
        double *o3) {
00248
00249
        double wp, wx, wv;
00250
00251
        int ip, ix, iy;
00252
        /* Check longitude... */
if (met->lon[met->nx - 1] > 180 && lon < 0)
00253
00254
          lon += 360;
00256
00257
        /* Get indices... */
00258
        ip = locate(met->p, met->np, p);
        ix = locate(met->lon, met->nx, lon);
00259
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
        /* Interpolate... */
00267
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);
        if (v != NULL)
00275
          intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
00276
        if (w != NULL)
          intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
00277
00278
        if (h2o != NULL)
00279
          intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
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(
       met_t * met0,
met_t * met1,
00288
00289
        double ts,
00290
        double p,
00291
        double lon.
00292
        double lat,
00293
        double *ps,
        double *t,
00294
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
        intpol_met_space(met0, p, lon, lat,
```

```
ps == NULL ? NULL : &ps0,
00306
                       t == NULL ? NULL : &t0,
00307
                       u == NULL ? NULL : &u0,
00308
                       v == NULL ? NULL : &v0,
00309
                       w == NULL ? NULL : &w0,
00310
                       h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
       00311
00312
00313
                       t == NULL ? NULL : &t1,
00314
                       u == NULL ? NULL : &u1,
                       v == NULL ? NULL : &v1,
00315
                       w == NULL ? NULL : &w1,
00316
00317
                       h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00318
00319
       /\star Get weighting factor... \star/
00320
       wt = (met1->time - ts) / (met1->time - met0->time);
00321
00322
       /* Interpolate... */
       if (ps != NULL)
00323
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)
00330
        *v = wt * (v0 - v1) + v1;
00331
       if (w != NULL)
       *w = wt * (w0 - w1) + w1;
if (h2o != NULL)
00332
00333
00334
        *h2o = wt * (h2o0 - h2o1) + h2o1;
       if (o3 != NULL)
00335
00336
        *o3 = wt * (o30 - o31) + o31;
00337 }
00338
00340
00341 void jsec2time(
00342
      double jsec,
00343
       int *year,
00344
       int *mon,
00345
       int *day,
00346
       int *hour,
00347
       int *min,
       int *sec,
00348
00349
       double *remain) {
00350
00351
       struct tm t0, *t1;
00352
00353
       time_t jsec0;
00354
00355
       t0.tm_year = 100;
00356
       t0.tm\_mon = 0;
00357
       t0.tm_mday = 1;
       t0.tm\_hour = 0;
00358
       t0.tm_min = 0;
00359
00360
       t0.tm sec = 0;
00361
00362
       jsec0 = (time_t) jsec + timegm(&t0);
00363
       t1 = gmtime(&jsec0);
00364
00365
       *year = t1->tm_year + 1900;
       *mon = t1->tm_mon + 1;
00366
00367
       *day = t1->tm_mday;
00368
       *hour = t1->tm_hour;
00369
       *min = t1->tm_min;
       *sec = t1->tm_sec;
00370
00371
       *remain = jsec - floor(jsec);
00372 }
00373
00375
00376 int locate(
       double *xx,
00377
00378
       int n,
00379
       double x) {
00380
00381
       int i, ilo, ihi;
00382
00383
       ilo = 0;
       ihi = n - 1;
00384
       i = (ihi + ilo) >> 1;
00385
00386
00387
       if (xx[i] < xx[i + 1])
00388
         while (ihi > ilo + 1) {
          i = (ihi + ilo) >> 1;
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;
if (xx[i] <= x)
00396
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
        char line[LEN], *tok;
00415
00416
00417
        int iq;
00418
00419
         /* Init... */
00420
        atm->np = 0;
00421
00422
        /* Write info... */
00423
        printf("Read atmospheric data: %s\n", filename);
00424
00425
        if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");
00426
00427
00428
        /* Read line... */
00430
        while (fgets(line, LEN, in)) {
00431
          /* Read data... */
TOK(line, tok, "%lg", atm->time[atm->np]);
TOK(NULL, tok, "%lg", atm->p[atm->np]);
TOK(NULL, tok, "%lg", atm->lon[atm->np]);
TOK(NULL, tok, "%lg", atm->lat[atm->np]);
for (iq = 0; iq < ctl->nq; iq++)
TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00432
00433
00434
00435
00436
00437
00438
00439
00440
           /* Convert altitude to pressure... */
00441
           atm \rightarrow p[atm \rightarrow np] = P(atm \rightarrow p[atm \rightarrow np]);
00442
           /* Increment data point counter... */
if ((++atm->np) > NP)
00443
00444
00445
             ERRMSG("Too many data points!");
00446
00447
00448
         /* Close file... */
00449
        fclose(in);
00450
00451
         /* Check number of points... */
        if (atm->np < 1)
00452
00453
          ERRMSG("Can not read any data!");
00454 }
00455
00457
00458 void read_ctl(
00459
        const char *filename,
00460
        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;
        ctl->qnt_r = -1;
00473
00474
        ctl->qnt_rho = -1;
        ctl->qnt_ps = -1;
00475
        ctl->qnt_p = -1;
00476
        ctl->qnt_t = -1;
ctl->qnt_u = -1;
00477
00478
```

```
ctl->qnt_v = -1;
        ctl->qnt_w = -1;
00480
00481
        ct1->qnt_h2o = -1;
        ctl->qnt_o3 = -1;
00482
00483
        ctl->qnt\_theta = -1;
00484
        ctl->ant pv = -1;
        ctl->qnt_tice = -1;
00486
        ctl->qnt_tnat = -1;
00487
        ctl->qnt\_stat = -1;
00488
00489
        /* Read quantities... */
00490
        ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
        for (iq = 0; iq < ctl->nq; iq++) {
00491
00492
00493
           /\star Read quantity name and format... \star/
          00494
00495
00496
00498
           /* Try to identify quantity... */
00499
          if (strcmp(ctl->qnt_name[iq], "m") == 0) {
00500
             ctl->qnt_m = iq;
             sprintf(ctl->qnt_unit[iq], "kg");
00501
          } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
00502
            ctl->qnt_r = iq;
00503
            sprintf(ctl->qnt_unit[iq], "m");
00504
00505
           } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
             ctl->qnt_rho = iq;
00506
00507
             sprintf(ctl->qnt_unit[iq], "kg/m^3");
          } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
  ctl->qnt_ps = iq;
00508
00509
00510
            sprintf(ctl->qnt_unit[iq], "hPa");
00511
          } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
00512
             ctl->qnt_p = iq;
00513
             sprintf(ctl->qnt_unit[iq], "hPa");
          } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
ctl->qnt_t = iq;
00514
00515
            sprintf(ctl->qnt_unit[iq], "K");
00517
          } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
00518
           ctl->qnt_u = iq;
00519
            sprintf(ctl->qnt_unit[iq], "m/s");
          } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
00520
            ctl->qnt_v = iq;
00521
            sprintf(ctl->qnt_unit[iq], "m/s");
          } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
00523
            ctl->qnt_w = iq;
00524
00525
            sprintf(ctl->qnt_unit[iq], "hPa/s");
          } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
  ctl->qnt_h2o = iq;
00526
00527
00528
            sprintf(ctl->qnt_unit[iq], "1");
          } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
00530
             ctl \rightarrow qnt_o3 = iq;
00531
             sprintf(ctl->qnt_unit[iq], "1");
00532
          } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
            ctl->qnt_theta = iq;
00533
            sprintf(ctl->qnt_unit[iq], "K");
00534
          } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
            ctl->qnt_pv = iq;
00536
00537
            sprintf(ctl->qnt_unit[iq], "PVU");
          } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
  ctl->qnt_tice = iq;
  sprintf(ctl->qnt_unit[iq], "K");
00538
00539
00540
          } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
00542
           ctl->qnt_tnat = iq;
00543
            sprintf(ctl->qnt_unit[iq], "K");
00544
          } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
            ctl->qnt_stat = iq;
00545
00546
            sprintf(ctl->qnt_unit[iq], "-");
00547
          } else
            scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00549
00550
00551
        /\star Time steps of simulation... \star/
00552
        ctl->direction =
          (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
f (ctl->direction != -1 && ctl->direction != 1)
00553
00554
00555
          ERRMSG("Set DIRECTION to -1 or 1!");
00556
        ctl->t_start =
        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);
00557
00558
00559
00561
         /* Meteorological data...
        ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
00562
        ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
00563
00564
        if (ctl->met_np > EP)
          ERRMSG("Too many levels!");
00565
```

```
for (ip = 0; ip < ctl->met_np; ip++)
00567
           ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
00568
00569
         /* Isosurface parameters... */
00570
         ctl->isosurf
         = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00571
00572
00573
00574
         /* Diffusion parameters... */
00575
         ctl->turb dx trop
           = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00576
00577
         ctl->turb dx strat
00578
            = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00579
         ctl->turb_dz_trop
00580
            = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00581
         ctl->turb_dz_strat
           = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00582
00583
         ctl->turb meso =
00584
           scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00585
00586
         /* Life time of particles... */
         ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
ctl->tdec_strat =
00587
00588
           scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00589
00590
         /* Output of atmospheric data... */
scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
00591
00592
      atm_basename);
00593
         scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00594
         ctl->atm_dt_out =
00595
           scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00596
         ctl->atm_filter
00597
           (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
00598
00599
         /* Output of CSI data... */
         scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
00600
      csi basename);
00601 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",
00602
00603
00604
                    ctl->csi_obsfile);
        ctl->csi_obsmin =
00605
           scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00606
         ctl->csi_modmin =
00607
         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);
00608
00609
00610
00611
         ctl->csi lon0 =
00612
00613
           scan_ctl(filename, argc, argv, "CSI_LONO", -1, "-180", NULL);
         ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
00614
00615
         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);
00616
00617
00618
00619
         ctl->csi nv =
00620
            (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
00621
00622
         /* Output of grid data... */
         scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
00623
00624
                    ctl->grid basename):
         scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
00625
      grid_gpfile);
00626
        ctl->grid_dt_out =
00627
           scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
         ctl->grid_sparse =
00628
         (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
00629
00630
00631
00632
         ctl->grid_nz =
00633
            (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00634
         ctl->grid lon0 =
00635
           scan_ctl(filename, argc, argv, "GRID_LONO", -1, "-180", NULL);
00636
         ctl->grid lon1
           scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
00637
00638
         ctl->grid nx =
00639
            (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00640
         ctl->grid_lat0 =
           scan_ctl(filename, argc, argv, "GRID_LATO", -1, "-90", NULL);
00641
00642
         ct1->grid lat1 =
           scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
00643
         ctl->grid_ny =
00644
00645
           (int) scan ctl(filename, argc, argv, "GRID NY", -1, "180", NULL);
00646
00647
         /\star Output of profile data... \star/
         00648
00649
```

```
00650
        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);
00651
00652
00653
        ctl->prof_nz =
           (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
00654
        ctl->prof_lon0 =
00656
           scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
00657
        ctl->prof_lon1
00658
          scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
        ctl->prof_nx =
00659
          (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00660
00661
        ctl->prof lat0
           scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
00662
00663
        ctl->prof_lat1 :
00664
          scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
        ctl->prof_ny =
00665
          (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00666
00667
00668
        /* Output of station data... */
00669
        scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
00670
                  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);
00671
00672
00673
00674 }
00675
00677
00678 void read met(
00679
        ctl_t * ctl,
00680
        char *filename,
        met_t * met) {
00681
00682
00683
        char tstr[10];
00684
00685
        static float help[EX * EY];
00686
00687
        int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00688
00689
        size_t np, nx, ny;
00690
        /* Write info... */
00691
00692
        printf("Read meteorological data: %s\n", filename);
00693
00694
        /\star Get time from filename... \star/
00695
        sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00696
        year = atoi(tstr);
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00697
00698
        mon = atoi(tstr);
        sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00699
00700
        day = atoi(tstr);
00701
         sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00702
        hour = atoi(tstr);
00703
        time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00704
00705
         /* Open netCDF file... */
00706
        NC(nc_open(filename, NC_NOWRITE, &ncid));
00707
        /* Get dimensions... */
NC(nc_inq_dimid(ncid, "lon", &dimid));
00708
00709
00710
        NC(nc_inq_dimlen(ncid, dimid, &nx));
00711
        if (nx > EX)
00712
          ERRMSG("Too many longitudes!");
00713
00714
        NC(nc_inq_dimid(ncid, "lat", &dimid));
00715
        NC(nc_inq_dimlen(ncid, dimid, &ny));
00716
        if (nv > EY)
00717
          ERRMSG("Too many latitudes!");
00718
00719
        NC(nc_inq_dimid(ncid, "lev", &dimid));
00720
        NC(nc_inq_dimlen(ncid, dimid, &np));
        if (np > EP)
00721
00722
          ERRMSG("Too many levels!");
00723
00724
        /* Store dimensions... */
        met->np = (int) np;
met->nx = (int) nx;
00725
00726
00727
        met->ny = (int) ny;
00728
00729
         /* Get horizontal grid... */
        NC(nc_inq_varid(ncid, "lon", &varid));
00730
        NC(nc_get_var_double(ncid, varid, met->lon));
NC(nc_inq_varid(ncid, "lat", &varid));
00731
00732
00733
        NC(nc_get_var_double(ncid, varid, met->lat));
00734
00735
        /* Read meteorological data... */
```

```
read_met_help(ncid, "t", "T", met, met->t, 1.0);
          read_met_help(ncid, "t", "l", 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, 0.01f);
read_met_help(ncid, "q", "Q", met, met->help(ncid, "q", "Q", met, met->help(ncid, "o3", "03", met, met->o3, 0.602f);
00737
00738
00739
00740
00741
00742
00743
          /\star Meteo data on pressure levels... \star/
00744
          if (ctl->met_np <= 0) {</pre>
00745
00746
             /* Read pressure levels from file... */
            NC(nc_inq_varid(ncid, "lev", &varid));
NC(nc_get_var_double(ncid, varid, met->p));
for (ip = 0; ip < met->np; ip++)
   met->p[ip] /= 100.;
00747
00748
00749
00750
00751
             /* Extrapolate data for lower boundary... */
00752
00753
            read_met_extrapolate(met);
00754
00755
00756
          /* Meteo data on model levels... */
          else {
00757
00758
00759
            /* Read pressure data from file... */
read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00760
00761
             /\star Interpolate from model levels to pressure levels... \star/
00762
00763
             read_met_ml2pl(ctl, met, met->t);
00764
             read_met_ml2pl(ctl, met, met->u);
00765
             read_met_ml2p1(ctl, met, met->v);
00766
             read_met_ml2pl(ctl, met, met->w);
00767
             read_met_ml2pl(ctl, met, met->h2o);
00768
             read_met_ml2pl(ctl, met, met->o3);
00769
00770
             /* Set pressure levels... */
00771
            met->np = ctl->met_np;
for (ip = 0; ip < met->np; ip++)
  met->p[ip] = ctl->met_p[ip];
00772
00773
00774
00775
00776
          /* Check ordering of pressure levels... */
00777
          for (ip = 1; ip < met->np; ip++)
  if (met->p[ip - 1] < met->p[ip])
00778
00779
               ERRMSG("Pressure levels must be descending!");
00780
00781
          /* Read surface pressure... */
00782
          if (nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
00783
            NC(nc_get_var_float(ncid, varid, help));
          for (iy = 0; iy < met->ny; iy++)
    for (ix = 0; ix < met->nx; ix++)
        met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
} else if (nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00784
00785
00786
00787
00788
            NC(nc_get_var_float(ncid, varid, help));
             for (iy = 0; iy < met->ny; iy++)
  for (ix = 0; ix < met->nx; ix++)
00789
00790
00791
                 met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00792
          } else
00793
            for (ix = 0; ix < met->nx; ix++)
00794
               for (iy = 0; iy < met->ny; iy++)
00795
                 met->ps[ix][iy] = met->p[0];
00796
00797
          /* Create periodic boundary conditions... */
00798
          read_met_periodic(met);
00799
00800
          /* Close file... */
00801
         NC(nc_close(ncid));
00802 }
00803
00805
00806 void read_met_extrapolate(
00807
         met_t * met) {
00808
00809
          int ip, ip0, ix, iy;
00810
00811
          /* Loop over columns... */
00812
          for (ix = 0; ix < met->nx; ix++)
00813
            for (iy = 0; iy < met->ny; iy++) {
00814
00815
               /* Find lowest valid data point... */
               for (ip0 = met->np - 1; ip0 >= 0; ip0--)
   if (!gsl_finite(met->t[ix][iy][ip0])
00816
00817
00818
                       || !gsl_finite(met->u[ix][iy][ip0])
00819
                       || !gsl_finite(met->v[ix][iy][ip0])
00820
                       || !gsl_finite(met->w[ix][iy][ip0]))
00821
                    break;
00822
```

```
/* Extrapolate... */
00824
           for (ip = ip0; ip >= 0; ip--) {
             met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
00825
             met \rightarrow u[ix][iy][ip] = met \rightarrow u[ix][iy][ip + 1];
00826
             met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
00827
             met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
00828
00830
             met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00831
00832
00833 }
00834
00836
00837 void read_met_help(
00838
       int ncid,
00839
       char *varname.
       char *varname2,
00840
00841
       met_t * met,
00842
       float dest[EX][EY][EP],
00843
       float scl) {
00844
00845
       static float help[EX * EY * EP];
00846
00847
       int ip, ix, iy, n = 0, varid;
00848
        /\star Check if variable exists... \star/
00849
00850
       if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00851
         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00852
           return:
00853
00854
        /* Read data... */
00855
       NC(nc_get_var_float(ncid, varid, help));
00856
00857
        /* Copy and check data... */
       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++];
00858
00859
00861
00862
              if (fabs(dest[ix][iy][ip] / scl) > 1e14)
00863
                dest[ix][iy][ip] = GSL_NAN;
00864
00865 }
00866
00868
00869 void read_met_ml2pl(
00870
       ctl_t * ctl,
       met_t * met,
00871
00872
       float var[EX][EY][EP]) {
00873
00874
       double aux[EP], p[EP], pt;
00875
00876
       int ip, ip2, ix, iy;
00877
00878
       /* Loop over columns... */
00879
       for (ix = 0; ix < met->nx; ix++)
00880
         for (iy = 0; iy < met->ny; iy++) {
00881
00882
            /\star Copy pressure profile... \star/
            for (ip = 0; ip < met->np; ip++)
00883
             p[ip] = met->pl[ix][iy][ip];
00884
00885
00886
            /* Interpolate... */
00887
            for (ip = 0; ip < ctl->met_np; ip++) {
00888
             pt = ctl->met_p[ip];
00889
              if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
               pt = p[0];
00890
00891
              else if ((pt > p[met->np - 1] && p[1] > p[0])
00892
                       || (pt < p[met->np - 1] && p[1] < p[0]))
00893
                pt = p[met->np - 1];
              ip2 = locate(p, met->np, pt);
00894
              00895
00896
00897
00898
00899
            /* Copy data... */
00900
            for (ip = 0; ip < ctl->met_np; ip++)
00901
             var[ix][iy][ip] = (float) aux[ip];
00902
00903 }
00904
00906
00907 void read_met_periodic(
00908
       met_t * met) {
00909
```

```
00910
         int ip, iy;
00911
00912
         /* Check longitudes... */
         if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
00913
00914
                      + met -> lon[1] - met -> lon[0] - 360) < 0.01))
00915
            return:
00916
00917
         /* Increase longitude counter... */
00918
         if ((++met->nx) > EX)
00919
           ERRMSG("Cannot create periodic boundary conditions!");
00920
00921
         /* Set longitude... */
00922
         met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
       lon[0];
00923
00924
          /* Loop over latitudes and pressure levels... */
         for (iy = 0; iy < met->ny; iy++)
  for (ip = 0; ip < met->np; ip++) {
    met->ps[met->nx - 1][iy] = met->ps[0][iy];
00925
00926
              met - u[met - nx - 1][iy][ip] = met - u[0][iy][ip];
00928
              met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
00929
00930
00931
              met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
00932
00933
00934
00935 }
00936
        ******************************
00937 /
00938
00939 double scan_ctl(
00940
         const char *filename,
00941
         int argc,
00942
         char *argv[],
00943
         const char *varname,
00944
         int arridx,
00945
         const char *defvalue,
00946
         char *value) {
00947
00948
         FILE *in = NULL;
00949
00950
         char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
00951
           msg[LEN], rvarname[LEN], rval[LEN];
00952
00953
         int contain = 0, i;
00954
00955
         /* Open file... */
         if (filename[strlen(filename) - 1] != '-')
if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");
00956
00957
00958
00959
00960
         /* Set full variable name... */
00961
         if (arridx >= 0) {
           sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
00962
00963
00964
          sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
00965
00966
00967
00968
00969
         /* Read data... */
00970
         if (in != NULL)
00971
           while (fgets(line, LEN, in))
00972
              if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
                if (strcasecmp(rvarname, fullname1) == 0 ||
00973
00974
                     strcasecmp(rvarname, fullname2) == 0) {
00975
                   contain = 1;
00976
                  break:
00977
         for (i = 1; i < argc - 1; i++)</pre>
          if (strcasecmp(argv[i], fullname1) == 0 ||
    strcasecmp(argv[i], fullname2) == 0) {
00979
00980
              sprintf(rval, "%s", argv[i + 1]);
00981
00982
              contain = 1;
00983
              break;
00984
00985
00986
         /\star Close file... \star/
         if (in != NULL)
00987
           fclose(in):
00988
00989
00990
         /\star Check for missing variables... \star/
         if (!contain) {
00991
00992
           if (strlen(defvalue) > 0)
00993
             sprintf(rval, "%s", defvalue);
00994
           else (
00995
             sprintf(msq, "Missing variable %s!\n", fullname1);
```

```
ERRMSG(msg);
00997
00998
00999
      /* Write info... */
printf("%s = %s\n", fullname1, rval);
01000
01001
01002
01003
       /* Return values... */
01004
       if (value != NULL)
        sprintf(value, "%s", rval);
01005
01006
      return atof(rval);
01007 }
01008
01010
01011 void time2jsec(
01012
      int year,
01013
       int mon,
01014
       int day,
01015
       int hour,
01016
       int min,
01017
       int sec,
01018
       double remain,
01019
       double *jsec) {
01020
01021
      struct tm t0, t1;
01022
01023
       t0.tm_year = 100;
       t0.tm_mon = 0;
01024
       t0.tm_mday = 1;
01025
       t0.tm_hour = 0;
01026
01027
       t0.tm_min = 0;
01028
       t0.tm_sec = 0;
01029
       t1.tm_year = year - 1900;
t1.tm_mon = mon - 1;
01030
01031
       t1.tm_mday = day;
01032
       t1.tm_hour = hour;
01033
01034
       t1.tm_min = min;
01035
       t1.tm_sec = sec;
01036
       *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01037
01038 }
01039
01041
01042 void timer(
01043
      const char *name,
01044
       int id.
01045
      int mode) {
01046
01047
       static double starttime[NTIMER], runtime[NTIMER];
01048
      /* Check id... */
if (id < 0 || id >= NTIMER)
01049
01050
01051
        ERRMSG("Too many timers!");
01052
01053
       /* Start timer... */
01054
       if (mode == 1) {
01055
        if (starttime[id] <= 0)</pre>
01056
          starttime[id] = omp_get_wtime();
01057
        else
01058
          ERRMSG("Timer already started!");
01059
01060
       /* Stop timer... */
else if (mode == 2) {
   if (starttime[id] > 0) {
01061
01062
01063
01064
          runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
          starttime[id] = -1;
01065
01066
01067
          ERRMSG("Timer not started!");
01068
      }
01069
      /* Print timer... */
else if (mode == 3)
01070
01071
01072
        printf("%s = %g s\n", name, runtime[id]);
01073 }
01074
01076
01077 double tropopause(
01078
     double t,
01079
       double lat) {
01080
01081
       static double doys[12]
01082
      = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
```

```
01084
                 static double lats[73]
                    = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
01085
                    -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5, -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5, -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
01086
01087
01088
                     15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
                     45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
01090
01091
                    75, 77.5, 80, 82.5, 85, 87.5, 90
01092
01093
01094
                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,
01095
01096
01097
                               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,
01098
                                                                                                                                                                    128,
01099
                               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,
01100
                               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,
01103
01104
                  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,
01105
01106
01107
                   220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8
01108
                  284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01109
01110
                  287.5, 286.2, 285.8},
                 {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9, 297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9, 161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3, 100.16, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 100.06, 
01111
01112
01113
                   100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01114
                   99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
01115
01116
                  186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
01117
                   279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1
                304.3, 304.9, 306, 306.6, 306.2, 306), {306.2, 306.7, 307.3, 306.4, 301.8, 290.2, 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,
01118
01119
01121
                   102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
01122
                  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, 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},
01123
01124
01125
01126
                 {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,
01127
01128
01129
                  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,
01130
01131
01132
                   273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
                   325.3, 325.8, 325.8},
01134
01135
                 {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,
01136
01137
                  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,
01138
                   127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
01140
                   251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
01141
                308.5, 312.2, 313.1, 313.3}, {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6, 187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6, 235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
01142
01143
01144
01145
                  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,
01147
01148
                  117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
                 224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5, 275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8, {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01149
01150
01151
                  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,
01153
01154
                  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,
01155
01156
                  230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7, 278.2, 282.6, 287.4, 290.9, 292.5, 293},
01157
                 {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
01159
01160
                   183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
01161
                  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, 110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
01162
01163
                   114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
                  203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5, 276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
01165
01166
                 {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5, 215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2, 237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
01167
01168
01169
```

```
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,
01171
01172
            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,
01173
01174
01175
            305.11.
           241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2, 253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
01176
01177
01178
            223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
01179
            108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
           102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2, 109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4, 241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6, 286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
01180
01181
01182
01183
01184
           {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,
01185
01186
01187
01188
01189
01190
            280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
01191
           281.7, 281.1, 281.2}
01192
          };
01193
01194
          double doy, p0, p1, pt;
01195
01196
          int imon, ilat;
01197
          /* Get day of year... */
doy = fmod(t / 86400., 365.25);
while (doy < 0)
01198
01199
01200
01201
            doy += 365.25;
01202
01203
          /* Get indices... */
          imon = locate(doys, 12, doy);
ilat = locate(lats, 73, lat);
01204
01205
01206
01207
           /* Get tropopause pressure... */
01208
          p0 = LIN(lats[ilat], tps[imon][ilat],
01209
                      lats[ilat + 1], tps[imon][ilat + 1], lat);
          01210
01211
          pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01212
01213
01214
          /* Return tropopause pressure... */
01215
          return pt;
01216 }
01217
01219
01220 void write_atm(
01221
          const char *filename,
01222
          ctl_t * ctl,
          atm_t * atm,
01223
01224
          double t) {
01225
01226
          FILE *in, *out;
01227
01228
          char line[LEN];
01229
01230
          double r, t0, t1;
01231
01232
          int ip, iq, year, mon, day, hour, min, sec;
01233
01234
          /* Set time interval for output... */
          t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01235
01236
01237
01238
          /* Check if gnuplot output is requested... */
          if (ctl->atm_gpfile[0] != '-') {
01240
01241
             /* Write info... */
01242
            printf("Plot atmospheric data: %s.png\n", filename);
01243
01244
            /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
01245
01246
               ERRMSG("Cannot create pipe to gnuplot!");
01247
             /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
01248
01249
01250
01251
             /* Set time string... */
             jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01252
01253
01254
                       year, mon, day, hour, min);
01255
01256
             /* Dump gnuplot file to pipe... */
```

```
if (!(in = fopen(ctl->atm_gpfile, "r")))
01258
           ERRMSG("Cannot open file!");
         while (fgets(line, LEN, in))
fprintf(out, "%s", line);
01259
01260
01261
         fclose(in);
01262
01263
01264
01265
         /* Write info... */ printf("Write atmospheric data: s\n", filename);
01266
01267
01268
01269
          /* Create file... */
01270
         if (!(out = fopen(filename, "w")))
01271
            ERRMSG("Cannot create file!");
01272
01273
01274
        /* Write header... */
       fprintf(out,
                "# $1 = time [s] \n"
01276
                "# \$2 = altitude [km]\n"
"# \$3 = longitude [deg]\n" "# \$4 = latitude [deg]\n");
01277
01278
       01279
01280
01281
01282
01283
01284
        /* Write data... */
        for (ip = 0; ip < atm->np; ip++) {
01285
01286
01287
          /* Check time... */
01288
          if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
01289
01290
         01291
01292
01293
          for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01294
01295
01296
            fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01297
01298
         fprintf(out, "\n");
01299
01300
01301
        /* Close file... */
01302
       fclose(out);
01303 }
01304
01306
01307 void write_csi(
01308
       const char *filename,
       ctl_t * ctl,
atm_t * atm,
01309
01310
01311
       double t) {
01312
01313
       static FILE *in, *out;
01314
01315
       static char line[LEN];
01316
       static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01317
01318
        rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01319
01320
       static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01321
        /* Init... */
01322
01323
        if (!init) {
01324
         init = 1;
01325
01326
          /\star Check quantity index for mass... \star/
01327
          if (ctl->qnt_m < 0)
01328
            ERRMSG("Need quantity mass to analyze CSI!");
01329
01330
          /* Open observation data file... */
         printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
if (!(in = fopen(ctl->csi_obsfile, "r")))
01331
01332
01333
            ERRMSG("Cannot open file!");
01334
         /* Create new file... */
printf("Write CSI data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01335
01336
01337
            ERRMSG("Cannot create file!");
01338
01339
01340
          /* Write header... */
          fprintf(out,
01341
                  "# $1 = time [s] \n"
01342
01343
                  "# $2 = number of hits (cx) \n"
```

```
"# $3 = number of misses (cy) \n"
                   "# $4 = number of false alarms (cz)\n"
01345
01346
                   "# $5 = number of observations (cx + cy) \n"
                   "# $6 = number of forecasts (cx + cz)\n"
01347
                    "# $7 = bias (forecasts/observations) [%%] \n'
01348
                   "# $8 = probability of detection (POD) [%%]\n"
"# $9 = false alarm rate (FAR) [%%]\n"
01349
01350
01351
                   "# $10 = critical success index (CSI) [%%]\n\n");
01352
01353
        /* Set time interval... */
01354
01355
        t0 = t - 0.5 * ct1 -> dt_mod;
        t1 = t + 0.5 * ct1 -> dt_mod;
01356
01357
01358
         /* Initialize grid cells... */
01359
        for (ix = 0; ix < ctl->csi_nx; ix++)
          for (iy = 0; iy < ctl->csi_ny; iy++)
  for (iz = 0; iz < ctl->csi_nz; iz++)
01360
01361
              modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01362
01363
01364
01365
        while (fgets(line, LEN, in)) {
01366
          /* Read data... */
if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01367
01368
01369
01370
01371
01372
          /* Check time... */
          if (rt < t0)</pre>
01373
01374
            continue;
01375
          if (rt > t1)
01376
            break;
01377
01378
          /* Calculate indices... */
          ix = (int) ((rlon - ctl->csi_lon0))
01379
          // (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
iy = (int) ((rlat - ctl->csi_lat0)
01380
01381
01382
                         (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01383
          iz = (int) ((rz - ctl -> csi_z0))
01384
                        / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01385
          /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
01386
01387
              iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01388
01389
01390
01391
          /* Get mean observation index... */
01392
          obsmean[ix][iy][iz] += robs;
01393
          obscount[ix][iy][iz]++;
01394
01395
01396
        /* Analyze model data... */
01397
        for (ip = 0; ip < atm->np; ip++) {
01398
01399
          /* Check time... */
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
01400
01401
            continue:
01402
01403
          /* Get indices... */
01404
          ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01405
                        / (ctl->csi lon1 - ctl->csi lon0) * ctl->csi nx);
01406
          iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
                        / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01407
          01408
01409
01410
          /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
01411
01412
              iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01413
01414
01415
01416
          /\star Get total mass in grid cell... \star/
          modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01417
01418
01419
01420
         /* Analyze all grid cells... */
01421
        for (ix = 0; ix < ctl->csi_nx; ix++)
          for (iy = 0; iy < ctl->csi_ny; iy++)
  for (iz = 0; iz < ctl->csi_nz; iz++) {
01422
01423
01424
01425
               /* Calculate mean observation index... */
               if (obscount[ix][iy][iz] > 0)
01426
01427
                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01428
               /\star Calculate column density... \star/
01429
01430
               if (modmean[ix][iy][iz] > 0) {
```

```
dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
                 dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
lat = ctl->csi_lat0 + dlat * (iy + 0.5);
01433
                 area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180. 
 * cos(lat * M_PI / 180.);
modmean[ix][iy][iz] /= (le6 * area);
01434
01435
01436
01437
01438
01439
               /* Calculate CSI... */
01440
               if (obscount[ix][iy][iz] > 0) {
                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01441
                      modmean[ix][iy][iz] >= ctl->csi_modmin)
01442
01443
                    cx++;
01444
                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01445
                            modmean[ix][iy][iz] < ctl->csi_modmin)
01446
                   cy++;
                 else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
01447
                            modmean[ix][iy][iz] >= ctl->csi_modmin)
01448
01449
                    cz++;
01450
               }
01451
01452
        /* Write output... */
01453
        if (fmod(t, ctl->csi_dt_out) == 0) {
01454
01455
           01456
01457
                    (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,

(cx + cz > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
01458
01459
01460
01461
01462
                    (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01463
01464
           /\star Set counters to zero... \star/
01465
          cx = cy = cz = 0;
01466
01467
01468
        /* Close file... */
01469
        if (t == ctl->t_stop)
01470
          fclose(out);
01471 }
01472
01474
01475 void write_grid(
01476
        const char *filename,
01477
        ctl_t * ctl,
        met_t * met0,
01478
01479
        met_t * met1,
01480
        atm_t * atm,
01481
        double t) {
01482
01483
        FILE *in, *out;
01484
        char line[LEN];
01485
01486
01487
        static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01488
          area, rho_air, press, temp, cd, mmr, t0, t1, r;
01489
01490
        static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01491
01492
        /* Check dimensions... */
        if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
    ERRMSG("Grid dimensions too large!");
01493
01494
01495
01496
         /\star Check quantity index for mass... \star/
01497
        if (ctl->qnt m < 0)
          ERRMSG("Need quantity mass to write grid data!");
01498
01499
         /* Set time interval for output... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01501
01502
01503
01504
         /* Set grid box size... */
        dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
01505
        dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
01506
01507
        dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
01508
        /* Initialize grid... */
for (ix = 0; ix < ctl->grid_nx; ix++)
   for (iy = 0; iy < ctl->grid_ny; iy++)
      for (iz = 0; iz < ctl->grid_nz; iz++)
01509
01510
01511
01512
01513
               grid_m[ix][iy][iz] = 0;
01514
01515
        /* Average data... */
        for (ip = 0; ip < atm->np; ip++)
  if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {</pre>
01516
01517
```

```
/* Get index... */
01519
              ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01520
01521
01522
01523
               /* Check indices... */
01525
               if (ix < 0 || ix >= ctl->grid_nx ||
01526
                   iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01527
                 continue;
01528
              /* Add mass... */
01529
              grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01530
01531
01532
         /* Check if gnuplot output is requested... */
if (ctl->grid_gpfile[0] != '-') {
01533
01534
01535
01536
            /* Write info... */
           printf("Plot grid data: %s.png\n", filename);
01537
01538
           /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
01539
01540
              ERRMSG("Cannot create pipe to gnuplot!");
01541
01542
            /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
01543
01544
01545
01546
            /* Set time string... */
            jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01547
01548
01549
                      year, mon, day, hour, min);
01550
01551
            /* Dump gnuplot file to pipe... */
           if (!(in = fopen(ctl->grid_gpfile, "r")))
    ERRMSG("Cannot open file!");
01552
01553
           while (fgets(line, LEN, in))
fprintf(out, "%s", line);
01554
01556
            fclose(in);
01557
01558
01559
         else {
01560
01561
            /* Write info... */
           printf("Write grid data: %s\n", filename);
01562
01563
            /* Create file... */
01564
           if (!(out = fopen(filename, "w")))
01565
              ERRMSG("Cannot create file!");
01566
01567
01568
01569
         /* Write header... */
01570
         fprintf(out,
01571
                   "# $1 = time [s] \n"
                   "# $2 = altitude [km] \n"
01572
01573
                   "# $3 = longitude [deg]\n"
01574
                   "# $4 = latitude [deg] \n"
01575
                   "# $5 = surface area [km^2]\n"
01576
                   "# $6 = layer width [km] \n"
                    "# $7 = temperature [K] \n"
01577
                   "# $8 = column density [kg/m^2]\n"
"# $9 = mass mixing ratio [1]\n\n");
01578
01579
01580
01581
         /* Write data... */
01582
         for (ix = 0; ix < ctl->grid_nx; ix++) {
01583
           if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
            fprintf(out, "\n");
for (iy = 0; iy < ctl->grid_ny; iy++) {
   if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
01584
01585
01586
               fprintf(out, "\n");
for (iz = 0; iz < ctl->grid_nz; iz++)
01588
01589
                 if (!ctl->grid_sparse
01590
                      | | ix == 0 | | iy == 0 | | iz == 0 | | grid_m[ix][iy][iz] > 0)  {
01591
01592
                   /* Set coordinates... */
01593
                   z = ctl->grid_z0 + dz * (iz + 0.5);
                   lon = ctl->grid_lon0 + dlon * (ix + 0.5);
lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01594
01595
01596
01597
                   /* Get pressure and temperature... */
                   press = P(z);
01598
                   intpol_met_time(met0, met1, t, press, lon, lat,
01600
                                       NULL, &temp, NULL, NULL, NULL, NULL, NULL);
01601
01602
                   /* Calculate surface area... */
                   area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
 * cos(lat * M_PI / 180.);
01603
01604
```

```
01606
                 /* Calculate column density... */
01607
                 cd = grid_m[ix][iy][iz] / (1e6 * area);
01608
                 /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01609
01610
01611
01612
                  /* Write output... */
01613
                 fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g,",
t, z, lon, lat, area, dz, temp, cd, mmr);
01614
01615
01616
01617
          }
01618
01619
01620
        /* Close file... */
01621
        fclose(out);
01622 }
01623
01625
01626 void write_prof(
01627
        const char *filename,
01628
        ctl_t * ctl,
met_t * met0,
01629
        met_t * met1,
01630
01631
        atm_t * atm,
01632
        double t) {
01633
01634
        static FILE *in, *out;
01635
01636
        static char line[LEN];
01637
01638
        static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
         rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, mmr, h2o, o3;
01639
01640
01641
01642
        static int init, obscount[GX][GY], ip, ix, iy, iz;
01643
01644
        /* Init... */
01645
        if (!init) {
01646
          init = 1;
01647
01648
           /* Check quantity index for mass... */
           if (ctl->qnt_m < 0)
01649
01650
             ERRMSG("Need quantity mass!");
01651
           /* Check dimensions... */
if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
    ERRMSG("Grid dimensions too large!");
01652
01653
01654
01655
01656
           /* Open observation data file... */
01657
           printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01658
           if (!(in = fopen(ctl->prof_obsfile, "r")))
             ERRMSG("Cannot open file!");
01659
01660
01661
           /* Create new file... */
01662
           printf("Write profile data: %s\n", filename);
01663
           if (!(out = fopen(filename, "w")))
             ERRMSG("Cannot create file!");
01664
01665
01666
           /* Write header... */
01667
           fprintf(out,
                    "# $1
                           = time [s]\n"
01668
01669
                    "# $2
                           = altitude [km] \n"
01670
                    "# $3 = longitude [deg] \n"
                    "# $4 = latitude [deg]\n"
01671
                    "# $5 = pressure [hPa]\n"
01672
01673
                    "# $6
                           = temperature [K]\n"
                    "# $7 = mass mixing ratio [1]\n"
01675
                    "# $8 = H20 volume mixing ratio [1]\n"
01676
                    "# $9 = 03 volume mixing ratio [1]\n"
                    "# $10 = mean BT index [K]\n");
01677
01678
01679
           /* Set grid box size... */
           dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
01680
          dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
01681
01682
01683
01684
        /* Set time interval... */
01685
        t0 = t - 0.5 * ct1->dt_mod;
01686
        t1 = t + 0.5 * ctl->dt_mod;
01687
01688
        /* Initialize... */
for (ix = 0; ix < ctl->prof_nx; ix++)
   for (iy = 0; iy < ctl->prof_ny; iy++) {
01689
01690
01691
```

```
obsmean[ix][iy] = 0;
             obscount[ix][iy] = 0;
01693
01694
             tmean[ix][iy] = 0;
             for (iz = 0; iz < ctl->prof_nz; iz++)
01695
01696
               mass[ix][iy][iz] = 0;
01697
          }
01698
01699
        /* Read data... */
01700
         while (fgets(line, LEN, in)) {
01701
01702
           /* Read data... */
01703
           if (sscanf(line, "%lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01704
            continue;
01705
01706
           /* Check time... */
01707
           if (rt < t0)
           continue;
if (rt > t1)
01708
01709
            break;
01710
01711
01712
           /* Calculate indices... */
          ix = (int) ((rlon - ctl->prof_lon0) / dlon);
iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01713
01714
01715
01716
           /* Check indices... */
01717
           if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01718
             continue;
01719
01720
           /\star Get mean observation index... \star/
           obsmean[ix][iy] += robs;
tmean[ix][iy] += rt;
01721
01722
01723
           obscount[ix][iy]++;
01724
01725
01726
         /* Analyze model data... */
         for (ip = 0; ip < atm->np; ip++) {
01727
01728
01729
           /* Check time... */
01730
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
01731
            continue;
01732
01733
           /* Get indices... */
          ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
01734
01735
01736
           iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01737
           /* Check indices... */
01738
           if (ix < 0 || ix >= ctl->prof_nx ||
    iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01739
01740
01741
             continue:
01742
01743
           /* Get total mass in grid cell... */
01744
           mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01745
01746
01747
         /* Extract profiles... */
01748
        for (ix = 0; ix < ctl->prof_nx; ix++)
01749
          for (iy = 0; iy < ctl->prof_ny; iy++)
01750
             if (obscount[ix][iy] > 0) {
01751
               /* Write output... */
01752
               fprintf(out, "\n");
01753
01754
01755
               /* Loop over altitudes... */
01756
               for (iz = 0; iz < ctl->prof_nz; iz++) {
01757
                 /* Set coordinates... */
01758
                 z = ctl - prof_z0 + dz * (iz + 0.5);
01759
                 lon = ctl - prof_lon0 + dlon * (ix + 0.5);
01760
                  lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01761
01762
01763
                  /\star Get meteorological data... \star/
01764
                  press = P(z);
                  intpol_met_time(met0, met1, t, press, lon, lat,
01765
                                    NULL, &temp, NULL, NULL, &h20, &o3);
01766
01767
01768
                  /* Calculate mass mixing ratio... */
                  rho_air = 100. * press / (287.058 * temp);
area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
* cos(lat * M_PI / 180.);
01769
01770
01771
01772
                  mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
01774
                  /* Write output... */
                  fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g,",
01775
01776
                           tmean[ix][iy] / obscount[ix][iy],
01777
                           z, lon, lat, press, temp, mmr, h2o, o3,
obsmean[ix][iy] / obscount[ix][iy]);
01778
```

```
01779
             }
01780
01781
01782
        /\star Close file... \star/
        if (t == ctl->t_stop)
01783
01784
         fclose(out);
01785 }
01786
01788
01789 void write station(
01790
       const char *filename.
       ctl_t * ctl,
atm_t * atm,
01791
01792
01793
       double t) {
01794
       static FILE *out:
01795
01796
01797
       static double rmax2, t0, t1, x0[3], x1[3];
01798
01799
       static int init, ip, iq;
01800
01801
       /* Init... */
       if (!init) {
01802
01803
         init = 1;
01804
          /* Write info... */
01805
01806
         printf("Write station data: %s\n", filename);
01807
01808
          /* Create new file... */
01809
         if (!(out = fopen(filename, "w")))
01810
           ERRMSG("Cannot create file!");
01811
01812
          /* Write header... */
01813
          fprintf(out,
                  "# $1 = time [s]\n"
01814
                  "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01815
01816
          01817
01818
01819
         fprintf(out, "\n");
01820
01821
01822
          /\star Set geolocation and search radius... \star/
01823
          geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
01824
          rmax2 = gsl_pow_2(ctl->stat_r);
01825
01826
       /* Set time interval for output... */
01827
       t0 = t - 0.5 * ct1->dt_mod;
01828
       t1 = t + 0.5 * ctl -> dt_mod;
01829
01830
01831
        /\star Loop over air parcels... \star/
01832
        for (ip = 0; ip < atm->np; ip++) {
01833
01834
          /* Check time... */
         if (atm->time[ip] < t0 || atm->time[ip] > t1)
01836
           continue;
01837
01838
          /* Check station flag... */
          if (ctl->qnt_stat >= 0)
01839
01840
           if (atm->q[ctl->qnt_stat][ip])
01841
              continue;
01842
01843
          /* Get Cartesian coordinates... */
01844
          geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
01845
01846
          /* Check horizontal distance... */
01847
         if (DIST2(x0, x1) > rmax2)
01848
           continue;
01849
01850
          /* Set station flag... */
01851
          if (ctl->qnt_stat >= 0)
01852
           atm->q[ctl->qnt_stat][ip] = 1;
01853
01854
          /* Write data... */
01855
          fprintf(out, "%.2f %g %g %g",
01856
                 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]);
01857
01858
01859
01860
01861
          fprintf(out, "\n");
01862
01863
        /* Close file... */
01864
01865
       if (t == ctl -> t stop)
```

```
01866 fclose(out);
01867 }
```

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 double radius;
00035 double radius;
00036 radius = NORM(x);
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.

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

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

Convert vertical distance to pressure.

Definition at line 92 of file libtrac.c.

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

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

Convert geolocation to Cartesian coordinates.

Definition at line 101 of file libtrac.c.

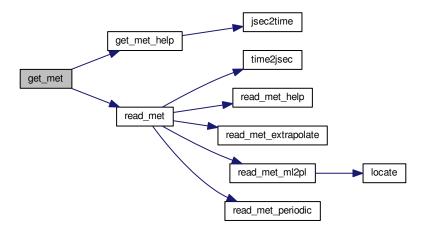
5.13.2.9 void get_met (ctl_t * ctl, char * metbase, double t, met_t * met0, met_t * met1)

Get meteorological data for given timestep.

Definition at line 117 of file libtrac.c.

```
00122
00123
00124
        char filename[LEN];
00125
00126
        static int init;
00127
00128
        /* Init... */
00129
        if (!init) {
00130
          init = 1;
00131
          get_met_help(t, -1, metbase, ctl->dt_met, filename);
00132
          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:



5.13.2.10 void get_met_help (double t, int direct, char * metbase, double dt_met, char * filename)

Get meteorological data for timestep.

Definition at line 156 of file libtrac.c.

```
00161
00162
00163
        double t6, r;
00164
00165
        int year, mon, day, hour, min, sec;
00166
        /* 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
        /* Set filename... */
sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", metbase, year, mon, day, hour);
00176
00177
00178 }
```

Here is the call graph for this function:



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

Linear interpolation of 2-D meteorological data.

Definition at line 182 of file libtrac.c.

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

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

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

Linear interpolation of 3-D meteorological data.

Definition at line 206 of file libtrac.c.

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

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

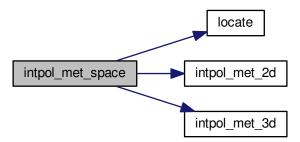
Spatial interpolation of meteorological data.

Definition at line 236 of file libtrac.c.

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

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

Here is the call graph for this function:



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

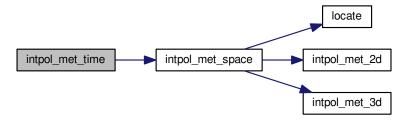
Temporal interpolation of meteorological data.

Definition at line 286 of file libtrac.c.

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

```
00319
        /* Get weighting factor... */
00320
       wt = (met1->time - ts) / (met1->time - met0->time);
00321
00322
        /\star Interpolate... \star/
00323
        if (ps != NULL)
        *ps = wt * (ps0 - ps1) + ps1;
if (t != NULL)
00324
00326
          *t = wt * (t0 - t1) + t1;
00327
        if (u != NULL)
        *u = wt * (u0 - u1) + u1;
if (v != NULL)
00328
00329
          *v = wt * (v0 - v1) + v1;
00330
        if (w != NULL)
00331
00332
          *w = wt * (w0 - w1) + w1;
        if (h2o != NULL)
00333
        *h2o = wt * (h2o0 - h2o1) + h2o1;
if (o3 != NULL)
00334
00335
          *o3 = wt * (o30 - o31) + o31;
00336
00337 }
```

Here is the call graph for this function:



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

Convert seconds to date.

Definition at line 341 of file libtrac.c.

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

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

Find array index.

Definition at line 376 of file libtrac.c.

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

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

Read atmospheric data.

Definition at line 408 of file libtrac.c.

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

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

Read control parameters.

Definition at line 458 of file libtrac.c.

```
00462
00463
00464
        int ip, iq;
00465
00466
        /* Write info... */
        printf("\nMassive-Parallel\ Trajectory\ Calculations\ (MPTRAC)\n"
00467
                 "(executable: %s | compiled: %s, %s)\n\n",
00468
                 argv[0], __DATE__, __TIME__);
00470
00471
        /* Initialize quantity indices... */
        ctl->qnt_m = -1;
ctl->qnt_r = -1;
00472
00473
00474
        ctl->qnt_rho = -1;
        ct1->qnt_ps = -1;
00475
        ctl->qnt_p = -1;
00476
         ctl->qnt_t = -1;
00477
         ctl->qnt_u = -1;
00478
        ctl \rightarrow qnt_v = -1;
00479
00480
        ctl->qnt w = -1;
00481
         ctl->qnt_h2o = -1;
00482
         ct1->qnt_o3 = -1;
00483
         ctl->qnt\_theta = -1;
00484
         ctl->qnt\_pv = -1;
00485
        ctl->qnt\_tice = -1;
        ctl \rightarrow qnt_tnat = -1;
00486
        ctl->qnt_stat = -1;
00487
00488
00489
         /* Read quantities... */
00490
         ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
         for (iq = 0; iq < ctl->nq; iq++) {
00491
00492
00493
           /* 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",
00494
00495
00496
                     ctl->qnt_format[iq]);
00497
           /* Try to identify quantity... */
if (strcmp(ctl->qnt_name[iq], "m") == 0) {
  ctl->qnt_m = iq;
00498
00499
00500
00501
             sprintf(ctl->qnt_unit[iq], "kg");
           } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
  ctl->qnt_r = iq;
00502
00503
             sprintf(ctl->qnt_unit[iq], "m");
00504
           } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
ctl->qnt_rho = iq;
sprintf(ctl->qnt_unit[iq], "kg/m^3");
00505
00506
00507
00508
           } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
00509
             ctl->qnt_ps = iq;
00510
             sprintf(ctl->qnt_unit[iq], "hPa");
           } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
00511
             ctl->qnt_p = iq;
00513
             sprintf(ctl->qnt_unit[iq], "hPa");
00514
           } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
00515
              ctl->qnt_t = iq;
00516
             sprintf(ctl->qnt_unit[iq], "K");
           } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
  ctl->qnt_u = iq;
00517
00518
             sprintf(ctl->qnt_unit[iq], "m/s");
00520
           } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
             ctl->qnt_v = iq;
00521
             sprintf(ctl->qnt_unit[iq], "m/s");
00522
           } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
  ctl->qnt_w = iq;
00523
00524
             sprintf(ctl->qnt_unit[iq], "hPa/s");
00525
           } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
ctl->qnt_h2o = iq;
00526
00527
             sprintf(ctl->qnt_unit[iq], "1");
00528
           } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
  ctl->qnt_o3 = iq;
  sprintf(ctl->qnt_unit[iq], "1");
00529
00530
00532
           } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
             ctl->qnt_theta = iq;
sprintf(ctl->qnt_unit[iq], "K");
00533
00534
           } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
  ctl->qnt_pv = iq;
00535
00536
00537
             sprintf(ctl->qnt_unit[iq], "PVU");
           } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
```

```
ctl->qnt_tice = iq;
00540
              sprintf(ctl->qnt_unit[iq], "K");
00541
            } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
             ctl->qnt_tnat = iq;
00542
              sprintf(ctl->qnt_unit[iq], "K");
00543
           } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
ctl->qnt_stat = iq;
00544
00546
              sprintf(ctl->qnt_unit[iq], "-");
00547
              scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00548
00549
00550
00551
         /* Time steps of simulation... */
00552
00553
           (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
         if (ctl->direction != -1 && ctl->direction != 1)
   ERRMSG("Set DIRECTION to -1 or 1!");
00554
00555
00556
         ctl->t start =
           scan_ctl(filename, argc, argv, "T_START", -1, "-1e100", 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);
00558
00559
00560
00561
         /* Meteorological data... */
         ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
00562
00563
         ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
         if (ctl->met_np > EP)
00564
00565
           ERRMSG("Too many levels!");
00566
         for (ip = 0; ip < ctl->met_np; ip++)
           ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
00567
00568
00569
         /* Isosurface parameters... */
00570
         ctl->isosurf
         = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00571
00572
00573
00574
         /* Diffusion parameters... */
00575
         ctl->turb dx trop
00576
            = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00577
         ctl->turb_dx_strat
00578
           = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00579
         ctl->turb_dz_trop
           = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00580
00581
         ctl->turb dz strat
00582
           = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00583
         ctl->turb meso =
00584
           scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00585
00586
         /\star Life time of particles... \star/
         ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
00587
         ctl->tdec_strat =
00588
           scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00590
00591
         /* Output of atmospheric data... */
00592
        scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
      atm_basename);
00593
         scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00594
         ctl->atm_dt_out =
00595
           scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00596
         ctl->atm_filter =
00597
            (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
00598
         /* Output of CSI data... */
00599
00600
         scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
00601
        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",
00602
00603
00604
                    ctl->csi_obsfile);
00605
         ctl->csi obsmin =
           scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00607
         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);
00608
00609
00610
00611
         ctl->csi_lon0 =
00612
00613
           scan_ctl(filename, argc, argv, "CSI_LONO", -1, "-180", NULL);
00614
         ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
         ctl->csi_nx =
00615
         (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);
00616
                                                                                      -90", NULL);
00617
00618
00619
         ctl->csi_ny =
00620
            (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
00621
00622
         /* Output of grid data... */
         scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
00623
```

```
ctl->grid_basename);
00624
         scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
00625
       grid_gpfile);
00626 ctl->grid_dt_out =
           scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
00627
00628
         ctl->grid sparse =
         (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
00630
00631
         ctl->grid_nz =
00632
           (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00633
00634
         ctl->grid lon0 =
00635
           scan_ctl(filename, argc, argv, "GRID_LONO", -1, "-180", NULL);
00636
         ctl->grid_lon1
00637
           scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
00638
         ctl->grid_nx =
           (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00639
00640
         ctl->grid lat0 =
00641
           scan_ctl(filename, argc, argv, "GRID_LATO", -1, "-90", NULL);
00642
         ctl->grid_lat1
            scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
00643
00644
         ctl->grid_ny =
           (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
00645
00646
00647
         /* Output of profile data... */
        scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
00648
00649
                    ctl->prof_basename);
00650
         scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
prof_obsfile);

00651 ctil=>=
         ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
00652
00653
         ctl->prof_nz =
00654
            (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
00655
         ctl->prof_lon0 =
00656
           scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
00657
         ctl->prof_lon1
00658
           scan ctl(filename, argc, argv, "PROF LON1", -1, "180", NULL);
         ctl->prof_nx =
00659
00660
            (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00661
         ctl->prof_lat0 =
00662
           scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
00663
         ctl->prof_lat1 =
           scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
00664
00665
         ctl->prof_ny =
00666
           (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00667
00668
         /* Output of station data... */
        scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
00669
00670
                   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);
00671
00672
00673
00674 }
```

Here is the call graph for this function:



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

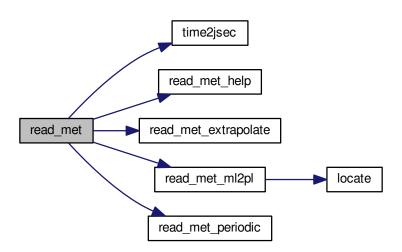
Read meteorological data file.

Definition at line 678 of file libtrac.c.

```
00681
00682
00683
         char tstr[10];
00684
00685
         static float help[EX * EY];
00686
00687
         int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00688
00689
         size_t np, nx, ny;
00690
00691
         /* Write info... */
         printf("Read meteorological data: %s\n", filename);
00692
00693
00694
          /* Get time from filename... */
00695
          sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
         year = atoi(tstr);
sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00696
00697
00698
         mon = atoi(tstr);
         sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00699
00700
         day = atoi(tstr);
00701
          sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00702
         hour = atoi(tstr);
00703
         time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00704
00705
           * Open netCDF file... */
00706
         NC(nc_open(filename, NC_NOWRITE, &ncid));
00707
         /* Get dimensions... */
NC(nc_inq_dimid(ncid, "lon", &dimid));
          /* Get dimensions...
00708
00709
00710
         NC(nc_inq_dimlen(ncid, dimid, &nx));
00711
         if (nx > EX)
00712
            ERRMSG("Too many longitudes!");
00713
00714
         NC(nc_inq_dimid(ncid, "lat", &dimid));
00715
         NC(nc_inq_dimlen(ncid, dimid, &ny));
00716
             (nv > EY)
00717
            ERRMSG("Too many latitudes!");
00718
00719
         NC(nc_inq_dimid(ncid, "lev", &dimid));
00720
         NC(nc_inq_dimlen(ncid, dimid, &np));
00721
          if (np > EP)
00722
           ERRMSG("Too many levels!");
00723
00724
         /* Store dimensions... */
00725
         met->np = (int) np;
00726
         met->nx = (int) nx;
00727
         met->ny = (int) ny;
00728
00729
          /* Get horizontal grid... */
         NC(nc_inq_varid(ncid, "lon", &varid));
00730
         NC(nc_get_var_double(ncid, varid, met->lon));
NC(nc_inq_varid(ncid, "lat", &varid));
00732
00733
         NC(nc_get_var_double(ncid, varid, met->lat));
00734
         /* 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);
00735
00736
00737
00738
00739
00740
00741
00742
00743
          /* Meteo data on pressure levels... */
00744
         if (ctl->met_np <= 0) {</pre>
00745
            /* Read pressure levels from file... */
NC(nc_inq_varid(ncid, "lev", &varid));
00746
00747
            NC(nc_get_var_double(ncid, varid, met->p));
for (ip = 0; ip < met->np; ip++)
  met->p[ip] /= 100.;
00748
00749
00750
00751
00752
            /\star Extrapolate data for lower boundary... \star/
00753
            read_met_extrapolate(met);
00754
00755
00756
          /* Meteo data on model levels... */
00757
00758
            /* Read pressure data from file... */
read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00759
00760
00761
00762
            /* Interpolate from model levels to pressure levels... */
00763
            read_met_ml2pl(ctl, met, met->t);
00764
            read_met_ml2pl(ctl, met, met->u);
00765
            read_met_ml2pl(ctl, met, met->v);
00766
            read_met_ml2pl(ctl, met, met->w);
00767
            read_met_ml2pl(ctl, met, met->h2o);
```

```
read_met_ml2pl(ctl, met, met->o3);
00769
00770
              /\star Set pressure levels... \star/
             met->np = ctl->met_np;
for (ip = 0; ip < met->np; ip++)
  met->p[ip] = ctl->met_p[ip];
00771
00772
00773
00774
00775
00776
           /\star Check ordering of pressure levels... \star/
           for (ip = 1; ip < met->np; ip++)
  if (met->p[ip - 1] < met->p[ip])
    ERRMSG("Pressure levels must be descending!");
00777
00778
00779
00780
           /* Read surface pressure... */
if (nc_ing_varid(ncid, "PS", &varid) == NC_NOERR) {
00781
00782
             NC(nc_get_var_float(ncid, varid, help));
00783
          for (iy = 0; iy < met->ny; iy++)
    for (ix = 0; ix < met->nx; ix++)
        met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
} else if (nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00784
00785
00786
00787
00788
             NC(nc_get_var_float(ncid, varid, help));
             for (iy = 0; iy < met->ny; iy++)
  for (ix = 0; ix < met->nx; ix++)
00789
00790
00791
                  met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00792
           } else
00793
             for (ix = 0; ix < met->nx; ix++)
00794
                for (iy = 0; iy < met->ny; iy++)
00795
                   met->ps[ix][iy] = met->p[0];
00796
00797
          /\star Create periodic boundary conditions... \star/
00798
          read_met_periodic(met);
00799
00800
           /* Close file... */
00801
           NC(nc_close(ncid));
00802 }
```

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 806 of file libtrac.c.

```
00807
00808
00809
         int ip, ip0, ix, iy;
00810
00811
         /* Loop over columns... */
00812
         for (ix = 0; ix < met->nx; ix++)
           for (iy = 0; iy < met->ny; iy++) {
00814
00815
               /* Find lowest valid data point... */
              for (ip0 = met->np - 1; ip0 >= 0; ip0--)
   if (!gsl_finite(met->t[ix][iy][ip0])
00816
00817
00818
                      | | !gsl_finite(met->u[ix][iy][ip0])
                      || !gsl_finite(met->v[ix][iy][ip0])
00819
00820
                      || !gsl_finite(met->w[ix][iy][ip0]))
00821
00822
00823
               /* Extrapolate... */
              /* Extrapolate... */
for (ip = ip0; ip >= 0; ip--) {
    met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
00824
00825
                 met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
00827
                met \rightarrow v[ix][iy][ip] = met \rightarrow v[ix][iy][ip + 1];
00828
                 met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
                met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00829
00830
00831
              }
00833 }
```

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 837 of file libtrac.c.

```
00844
00845
        static float help[EX * EY * EP];
00846
00847
        int ip, ix, iv, n = 0, varid;
00848
00849
         /* Check if variable exists... */
00850
         if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00851
         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00852
             return:
00853
         /* Read data...
00854
00855
        NC(nc_get_var_float(ncid, varid, help));
00856
00857
         /* Copy and check data... */
        for (ip = 0; ip < met->np; ip++)
  for (iy = 0; iy < met->ny; iy++)
00858
00859
             for (ix = 0; ix < met->nx; ix++) {
  dest[ix][iy][ip] = scl * help[n++];
00860
00862
               if (fabs(dest[ix][iy][ip] / scl) > 1e14)
00863
                  dest[ix][iy][ip] = GSL_NAN;
00864
             }
00865 }
```

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 869 of file libtrac.c.

```
00872 {
00873
00874 double aux[EP], p[EP], pt;
00875
00876 int ip, ip2, ix, iy;
00877
00878 /* Loop over columns... */
00879 for (ix = 0; ix < met->nx; ix++)
00880 for (iy = 0; iy < met->ny; iy++) {
```

```
/* Copy pressure profile... */
              for (ip = 0; ip < met->np; ip++)
  p[ip] = met->pl[ix][iy][ip];
00883
00884
00885
              /* Interpolate... */
for (ip = 0; ip < ctl->met_np; ip++) {
   pt = ctl->met_p[ip];
00886
00887
00888
00889
                if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
00890
                  pt = p[0];
                else if ((pt > p[met->np - 1] && p[1] > p[0])
  || (pt < p[met->np - 1] && p[1] < p[0]))
00891
00892
                  pt = p[met->np - 1];
00893
                00894
00895
00896
00897
00898
00899
              /* Copy data... */
for (ip = 0; ip < ctl->met_np; ip++)
00900
00901
                var[ix][iy][ip] = (float) aux[ip];
00902
00903 }
```

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 907 of file libtrac.c.

```
00908
00909
00910
          int ip, iy;
00912
          /* Check longitudes... */
00913
          if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
00914
                          + met - > lon[1] - met - > lon[0] - 360) < 0.01))
00915
             return:
00916
00917
          /* Increase longitude counter... */
00918
          if ((++met->nx) > EX)
00919
             ERRMSG("Cannot create periodic boundary conditions!");
00920
         /* Set longitude... */
met->lon[met->nx - 2] + met->lon[1] - met->
00921
00922
        lon[0];
00923
00924
           /* Loop over latitudes and pressure levels... */
00925
           for (iy = 0; iy < met->ny; iy++)
             for (ip = 0; ip < met >ny, 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];
00926
00927
00928
00929
                met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
00930
00931
               met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
00932
00933
00934
00935 }
```

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 939 of file libtrac.c.

```
00946
00947
00948
        FILE *in = NULL;
00949
00950
        char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
          msg[LEN], rvarname[LEN], rval[LEN];
00951
00952
00953
        int contain = 0, i;
00954
00955
         /* Open file... */
        if (filename[strlen(filename) - 1] != '-')
if (!(in = fopen(filename, "r")))
00956
00957
             ERRMSG("Cannot open file!");
00959
00960
         /* Set full variable name... */
         if (arridx >= 0) {
00961
         sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
00962
00963
00964
         sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
00965
00966
00967
00968
00969
        /* Read data... */
        if (in != NULL)
00970
         while (fgets(line, LEN, in))
00971
00972
            if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
00973
               if (strcasecmp(rvarname, fullname1) == 0 | |
00974
                    strcasecmp(rvarname, fullname2) == 0) {
00975
                  contain = 1;
00976
                 break;
00977
               }
00978
        for (i = 1; i < argc - 1; i++)</pre>
        if (strcasecmp(argv[i], fullname1) == 0 ||
    strcasecmp(argv[i], fullname2) == 0) {
    sprintf(rval, "%s", argv[i + 1]);
00979
00980
00981
00982
            contain = 1;
00983
             break;
00984
00985
00986
        /* Close file... */
if (in != NULL)
00987
00988
          fclose(in);
00990
        /* Check for missing variables... */
00991
        if (!contain) {
         if (strlen(defvalue) > 0)
00992
00993
             sprintf(rval, "%s", defvalue);
00994
          else {
00995
            sprintf(msg, "Missing variable %s!\n", fullname1);
00996
             ERRMSG(msg);
00997
00998
00999
        /* Write info... */
01000
        printf("%s = %s\n", fullname1, rval);
01001
01003
         /* Return values... */
01004
        if (value != NULL)
01005
          sprintf(value, "%s", rval);
01006
        return atof(rval);
01007 }
```

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 1011 of file libtrac.c.

```
01020
01021
       struct tm t0, t1;
01022
       t0.tm_year = 100;
01023
01024
       t0.tm mon = 0;
       t0.tm_mday = 1;
01025
01026
        t0.tm\_hour = 0;
01027
       t0.tm_min = 0;
01028
       t0.tm_sec = 0;
01029
       t1.tm_year = year - 1900;
01030
       t1.tm_mon = mon - 1;
01031
01032
       t1.tm_mday = day;
01033
       t1.tm_hour = hour;
01034
       t1.tm_min = min;
       t1.tm_sec = sec;
01035
01036
       *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01038 }
```

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

Measure wall-clock time.

Definition at line 1042 of file libtrac.c.

```
01045
                   {
01046
01047
        static double starttime[NTIMER], runtime[NTIMER];
01048
       /* Check id... */
if (id < 0 || id >= NTIMER)
01049
01050
01051
          ERRMSG("Too many timers!");
01052
01053
        /* Start timer... */
        if (mode == 1) {
01054
         if (starttime[id] <= 0)</pre>
01055
01056
           starttime[id] = omp_get_wtime();
          else
01058
            ERRMSG("Timer already started!");
01059
01060
        /* Stop timer... */
else if (mode == 2) {
01061
01062
01063
        if (starttime[id] > 0) {
01064
            runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
01065
            starttime[id] = -1;
01066
            ERRMSG("Timer not started!");
01067
01068
01069
01070
       /* Print timer... */
01071
        else if (mode == 3)
          printf("%s = %g s\n", name, runtime[id]);
01072
01073 }
```

5.13.2.27 double tropopause (double t, double lat)

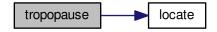
Definition at line 1077 of file libtrac.c.

```
01081
                   static double doys[12]
                   = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01082
01083
                   static double lats[73]
01084
                      static double lats[73]
= { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5, -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5, -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5, -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5, 15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5, 45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5, 75, 77.5, 80, 82.5, 85, 87.5, 90
01085
01086
01087
01088
01089
01090
01091
01092
```

```
01094
             static double tps[12][73]
01095
                = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
01096
                         297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
01097
                         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,
01098
                         152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
01100
01101
                         277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
             275.3, 275.6, 275.4, 274.1, 273.5}, {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7, 300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
01102
01103
01104
               150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
01105
               98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
01106
01107
               98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
01108
               220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
               284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01109
               287.5, 286.2, 285.8},
01110
             {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
01111
               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,
01113
01114
               100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01115
               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,
01116
01117
               304.3, 304.9, 306, 306.6, 306.2, 306},
01118
              {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
01119
01120
               290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
               195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7, 102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79, 99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
01121
01122
01123
01124
               148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6,
               263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.
01125
01126
               315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
             {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8, 260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9, 205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7, 101.8, 101.4, 101.1, 101, 101, 101.1, 101.2, 101.5, 101.9,
01127
01128
01129
01131
               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,
01132
01133
               273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
               325.3, 325.8, 325.8},
01134
             323.3, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 323.8, 32
01135
01136
01138
01139
               106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
01140
               127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
             251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9, 308.5, 312.2, 313.1, 313.3}, {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
01141
01142
01143
               187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
01144
01145
               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,
01146
01147
               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,
01148
               275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
01150
             {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01151
01152
               185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
01153
               233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
              110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.6, 111.9, 112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
01154
               120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
01156
01157
               230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7
01158
               278.2, 282.6, 287.4, 290.9, 292.5, 293},
01159
             {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
               183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
01160
               243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
01161
               114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4,
               110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
01163
01164
               114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
               203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5, 276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
01165
01166
             {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,
01167
               237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7,
01169
                                                                                                  124.8,
01170
               111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01171
               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,
01172
               206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01173
               279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
               305.1},
01175
              {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01176
01177
               253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
               223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9, 108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
01178
```

```
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,
01182
            286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6}, {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7, 284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192, 175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1, 100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
01183
01184
01185
01186
01187
             100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160, 186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278, 280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4, 281.7, 281.1, 281.2}
01188
01189
01190
01191
01192
01193
01194
            double doy, p0, p1, pt;
01195
01196
            int imon, ilat:
01197
           /* Get day of year... */
01198
            doy = fmod(t / 86400., 365.25);
01199
01200
            while (doy < 0)
01201
               doy += 365.25;
01202
01203
            /* Get indices... */
           imon = locate(doys, 12, doy);
ilat = locate(lats, 73, lat);
01204
01205
01206
01207
             /* Get tropopause pressure... */
           01208
01209
            p1 = LIN(lats[ilat], tps[imon + 1][ilat],
lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
01210
01211
01212
            pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01213
01214
             /* Return tropopause pressure... */
01215
            return pt;
01216 }
```

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 1220 of file libtrac.c.

```
01224
01225
01226
        FILE *in, *out;
01227
01228
        char line[LEN];
01229
01230
        double r, t0, t1;
01231
01232
        int ip, iq, year, mon, day, hour, min, sec;
01233
01234
         /* Set time interval for output... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01235
01236
01237
01238
        /* Check if gnuplot output is requested... */
        if (ctl->atm_gpfile[0] != '-') {
```

```
01240
01241
           /* Write info... */
01242
           printf("Plot atmospheric data: %s.png\n", filename);
01243
01244
          /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
01245
01246
             ERRMSG("Cannot create pipe to gnuplot!");
01247
          /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
01248
01249
01250
01251
           /* Set time string... */
           jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01252
01253
01254
                    year, mon, day, hour, min);
01255
           /* Dump gnuplot file to pipe... */
01256
          if (!(in = fopen(ctl->atm_gpfile, "r")))
ERRMSG("Cannot open file!");
01257
01258
01259
          while (fgets(line, LEN, in))
             fprintf(out, "%s", line);
01260
01261
          fclose(in);
01262
01263
01264
        else {
01265
01266
           /\star Write info... \star/
01267
          printf("Write atmospheric data: %s\n", filename);
01268
01269
           /* Create file... */
          if (!(out = fopen(filename, "w")))
01270
01271
             ERRMSG("Cannot create file!");
01272
01273
01274
         /* Write header... */
        01275
01276
                 "# $2 = altitude [km] \n"
01278
                 "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
        01279
01280
01281
01282
01283
01284
         /* Write data... */
01285
         for (ip = 0; ip < atm->np; ip++) {
01286
          /* Check time... */
if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
01287
01288
01289
             continue:
01290
01291
           /* Write output... */
          fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
    atm->lon[ip], atm->lat[ip]);
01292
01293
           for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01294
01295
01296
01297
           fprintf(out, "\n");
01298
01299
01300
        /* Close file... */
01301
01302
        fclose(out);
01303 }
```

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 1307 of file libtrac.c.

```
01311
                    {
01312
01313
        static FILE *in, *out;
01314
01315
        static char line[LEN]:
01316
        static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01317
01318
          rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01319
01320
        static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01321
01322
         /* Init... */
01323
        if (!init) {
01324
          init = 1;
01325
01326
           /* Check quantity index for mass... */
01327
           if (ctl->qnt_m < 0)
             ERRMSG("Need quantity mass to analyze CSI!");
01328
01329
01330
           /* Open observation data file... */
01331
           printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
01332
           if (!(in = fopen(ctl->csi_obsfile, "r")))
             ERRMSG("Cannot open file!");
01333
01334
01335
           /* Create new file... */
          /* Create new IIIe... ^/
printf("Write CSI data: %s\n", filename);
if (!/out = fonen(filename, "w")))
01336
01337
01338
            ERRMSG("Cannot create file!");
01339
01340
           /* Write header... */
01341
          fprintf(out,
                    "# $1 = time [s]\n"
01342
01343
                   "# $2 = number of hits (cx) \n"
01344
                    "# $3 = number of misses (cy) \n"
01345
                    "# $4 = number of false alarms (cz)\n"
                    "# $5 = number of observations (cx + cy) \n"
01346
                    "# $6 = number of forecasts (cx + cz) n"
01347
                    "# $7 = bias (forecasts/observations) [%%]\n"
01348
                   "# $8 = probability of detection (POD) [%%]\n"
"# $9 = false alarm rate (FAR) [%%]\n"
01349
01350
01351
                    "# $10 = critical success index (CSI) [%%]\n\n");
01352
01353
01354
        /* Set time interval... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01355
01356
01357
        /\star Initialize grid cells... \star/
01358
        for (ix = 0; ix < ctl->csi_nx; ix++)
  for (iy = 0; iy < ctl->csi_ny; iy++)
    for (iz = 0; iz < ctl->csi_nz; iz++)
01359
01360
01361
01362
               modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01363
01364
         /* Read data... */
01365
        while (fgets(line, LEN, in)) {
01366
01367
           /* Read data... *
           if (sscanf(line, "%lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01368
01369
01370
             continue;
01371
           /* Check time... */
01372
01373
           if (rt < t0)
01374
             continue;
01375
           if (rt > t1)
01376
            break;
01377
01378
           /* Calculate indices... */
01379
          ix = (int) ((rlon - ctl->csi_lon0)
                         / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01380
01381
           iy = (int) ((rlat - ctl -> csi_lat0))
01382
                         / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
           iz = (int) ((rz - ctl -> csi_z0)
01383
01384
                        / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01385
01386
           /* Check indices... */
01387
           if (ix < 0 || ix >= ctl->csi_nx ||
```

```
iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01389
01390
01391
           /* Get mean observation index... */
01392
          obsmean[ix][iy][iz] += robs;
          obscount[ix][iy][iz]++;
01393
01394
01395
01396
        /* Analyze model data... ∗/
01397
        for (ip = 0; ip < atm->np; ip++) {
01398
01399
          /* Check time... */
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
01400
01401
            continue;
01402
          /* Get indices... */
ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01403
01404
                        / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01405
          iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
01406
                        / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01407
          01408
01409
01410
          /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
01411
01412
              iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01413
01414
01415
01416
          /* Get total mass in grid cell... */
          modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01417
01418
01419
01420
        /* Analyze all grid cells... */
01421
        for (ix = 0; ix < ctl->csi_nx; ix++)
         for (iy = 0; iy < ctl->csi_ny; iy++)
  for (iz = 0; iz < ctl->csi_nz; iz++) {
01422
01423
01424
               /\star Calculate mean observation index... \star/
01426
               if (obscount[ix][iy][iz] > 0)
01427
                obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01428
01429
               /* Calculate column density... */
              if (modmean[ix][iy][iz] > 0) {
  dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
  dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
01430
01431
01432
01433
                 lat = ctl->csi\_lat0 + dlat * (iy + 0.5);
                area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180. * cos(lat * M_PI / 180.);
01434
01435
                modmean[ix][iy][iz] /= (1e6 * area);
01436
01437
01438
01439
               /* Calculate CSI... */
01440
               if (obscount[ix][iy][iz] > 0) {
01441
               if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
                     modmean[ix][iy][iz] >= ctl->csi_modmin)
01442
01443
                   cx++;
                else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01445
                          modmean[ix][iy][iz] < ctl->csi_modmin)
01446
                 01447
01448
01449
                   cz++;
01450
              }
01451
01452
01453
        /* Write output... */
01454
        if (fmod(t, ctl->csi_dt_out) == 0) {
01455
          /* Write... */
fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
01456
01457
                  (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN, (cx + cz > 0) ? (100. * cx) / (cx + cz) : GSL_NAN,
01458
01459
01460
01461
                   (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01462
01463
01464
          /* Set counters to zero... */
01465
          cx = cy = cz = 0;
01466
01467
        /* Close file... */
01468
        if (t == ctl->t_stop)
01469
01470
          fclose(out);
01471 }
```

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

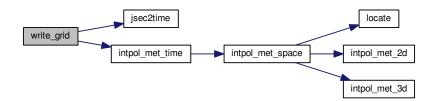
Write gridded data.

Definition at line 1475 of file libtrac.c.

```
01481
                      {
01482
01483
         FILE *in, *out;
01484
01485
         char line[LEN];
01486
01487
         static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
            area, rho_air, press, temp, cd, mmr, t0, t1, r;
01489
01490
         static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01491
         /* Check dimensions... */
if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
    ERRMSG("Grid dimensions too large!");
01492
01493
01494
01495
01496
          /\star Check quantity index for mass... \star/
01497
         if (ctl->qnt_m < 0)
           ERRMSG("Need quantity mass to write grid data!");
01498
01499
01500
         /* Set time interval for output... */
         t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01501
01502
01503
01504
          /* Set grid box size... */
         dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
01505
         dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
01506
01508
01509
          /* Initialize grid... */
         for (ix = 0; ix < ctl->grid_nx; ix++)
  for (iy = 0; iy < ctl->grid_ny; iy++)
    for (iz = 0; iz < ctl->grid_nz; iz++)
01510
01511
01512
01513
                 grid_m[ix][iy][iz] = 0;
01515
          /* Average data... */
         for (ip = 0; ip < atm->np; ip++)
  if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {</pre>
01516
01517
01518
               /* Get index... */
              ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
01520
01521
01522
              iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01523
01524
               /* Check indices... */
              if (ix < 0 || ix >= ctl->grid_nx ||
01525
                   iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01526
01527
01528
              /* Add mass... */
grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01529
01530
01531
01532
01533
          /* Check if gnuplot output is requested... */
01534
         if (ctl->grid_gpfile[0] != '-') {
01535
            /* Write info... */
01536
01537
            printf("Plot grid data: %s.png\n", filename);
01539
             /* Create gnuplot pipe... */
            if (!(out = popen("gnuplot", "w")))
01540
              ERRMSG("Cannot create pipe to gnuplot!");
01541
01542
01543
            /* Set plot filename... */
            fprintf(out, "set out \"%s.png\"\n", filename);
01544
01545
            /* Set time string... */
01546
            jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01547
01548
01549
                      year, mon, day, hour, min);
01550
01551
            /\star Dump gnuplot file to pipe... \star/
            if (!(in = fopen(ctl->grid_gpfile, "r")))
    ERRMSG("Cannot open file!");
01552
01553
            while (fgets(line, LEN, in))
  fprintf(out, "%s", line);
01554
01555
01556
            fclose(in);
01557
```

```
01558
01559
01560
01561
           /* Write info... */
01562
           printf("Write grid data: %s\n", filename);
01563
01564
            /* Create file... */
01565
            if (!(out = fopen(filename, "w")))
              ERRMSG("Cannot create file!");
01566
01567
01568
         /* Write header... */
01569
01570
         fprintf(out,
01571
                   "# $1 = time [s] \n"
                   "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n"
"# $4 = latitude [deg]\n"
01572
01573
01574
                   "# $5 = surface area [km^2]\n"
"# $6 = layer width [km]\n"
01575
01576
01577
                   "# $7 = temperature [K]\n"
01578
                   "# $8 = \text{column density } [kg/m^2] \n"
                   "# $9 = mass mixing ratio [1] n n");
01579
01580
         /* Write data... */
for (ix = 0; ix < ctl->grid_nx; ix++) {
   if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01581
01582
01584
              fprintf(out, "\n");
            for (iy = 0; iy < ctl->grid_ny; iy++) {
   if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
01585
01586
              fprintf(out, "\n");
for (iz = 0; iz < ctl->grid_nz; iz++)
01587
01588
01589
                if (!ctl->grid_sparse
01590
                      || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01591
                   /* Set coordinates... */
z = ctl->grid_z0 + dz * (iz + 0.5);
lon = ctl->grid_lon0 + dlon * (ix + 0.5);
01592
01593
01594
                   lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01595
01596
01597
                   /* Get pressure and temperature... */
01598
                   press = P(z);
                   01599
01600
01601
01602
                   /* Calculate surface area... */
                   area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
 * cos(lat * M_PI / 180.);
01603
01604
01605
01606
                   /* Calculate column density... */
                   cd = grid_m[ix][iy][iz] / (1e6 * area);
01607
01608
                  /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01609
01610
01611
01612
                    /* Write output...
01613
                   fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
01614
01615
                             t, z, lon, lat, area, dz, temp, cd, mmr);
01616
01617
           }
01618
         }
01619
01620
          /* Close file... */
01621
         fclose(out);
01622 }
```

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)

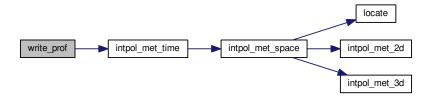
Write profile data.

Definition at line 1626 of file libtrac.c.

```
01632
                      {
01633
01634
         static FILE *in, *out;
01635
01636
         static char line[LEN]:
01637
         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,
01638
01639
01640
           press, temp, rho_air, mmr, h2o, o3;
01641
01642
         static int init, obscount[GX][GY], ip, ix, iy, iz;
01643
01644
         /* Init... */
01645
         if (!init) {
01646
           init = 1;
01647
01648
            /\star Check quantity index for mass... \star/
            if (ctl->qnt_m < 0)</pre>
01649
              ERRMSG("Need quantity mass!");
01650
01651
01652
            /* Check dimensions... */
01653
             \begin{tabular}{ll} if & (ctl->prof_nx > GX \ || \ ctl->prof_ny > GY \ || \ ctl->prof_nz > GZ) \end{tabular} 
01654
              ERRMSG("Grid dimensions too large!");
01655
01656
            /* Open observation data file... */
            printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01657
               (!(in = fopen(ctl->prof_obsfile, "r")))
01659
             ERRMSG("Cannot open file!");
01660
           /* Create new file... */
printf("Write profile data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01661
01662
01663
              ERRMSG("Cannot create file!");
01664
01665
01666
            /* Write header... */
01667
            fprintf(out,
                      "# $1
                             = time [s]\n"
01668
                      "# $2
                             = altitude [km]\n"
01669
                     "# $3
                             = longitude [deg]\n"
01670
01671
                     "# $4 = latitude [deg]\n"
01672
                      "# $5
                             = pressure [hPa]\n"
01673
                      "# $6 = temperature [K]\n"
                      "# $7
01674
                             = mass mixing ratio [1]\n"
                     "# $8 = H2O volume mixing ratio [1]\n"
"# $9 = 03 volume mixing ratio [1]\n"
01675
01676
01677
                     "# $10 = mean BT index [K]\n");
01678
01679
            /* Set grid box size... */
           dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
01680
01681
01682
01683
01684
01685
         /\star Set time interval... \star/
         t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01686
01687
01688
01689
          /* Initialize... */
01690
         for (ix = 0; ix < ctl->prof_nx; ix++)
01691
           for (iy = 0; iy < ctl->prof_ny; iy++) {
              obsmean[ix][iy] = 0;
01692
              obscount[ix][iy] = 0;
01693
              tmean[ix][iy] = 0;
for (iz = 0; iz < ctl->prof_nz; iz++)
01694
01695
                 mass[ix][iy][iz] = 0;
01696
01697
01698
         /* Read data... */
while (fgets(line, LEN, in)) {
01699
01700
01701
01702
01703
            if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01704
              continue;
01705
01706
            /* Check time... */
01707
           if (rt < t0)</pre>
01708
              continue;
```

```
01709
           if (rt > t1)
01710
             break;
01711
           /* Calculate indices... */
ix = (int) ((rlon - ctl->prof_lon0) / dlon);
iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01712
01713
01714
01715
01716
            /* Check indices... */
01717
           if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01718
             continue;
01719
01720
           /* Get mean observation index... */
           obsmean[ix][iy] += robs;
tmean[ix][iy] += rt;
01721
01722
01723
           obscount[ix][iy]++;
01724
01725
01726
         /* Analyze model data... */
         for (ip = 0; ip < atm->np; ip++) {
01728
01729
            /* Check time... */
01730
           if (atm->time[ip] < t0 || atm->time[ip] > t1)
01731
             continue;
01732
01733
           /* Get indices... */
01734
           ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
           iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01735
01736
01737
           /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx ||
01738
01739
                iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01741
01742
           /* Get total mass in grid cell... */
mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01743
01744
01745
01746
01747
         /* Extract profiles... */
01748
         for (ix = 0; ix < ctl->prof_nx; ix++)
01749
           for (iy = 0; iy < ctl->prof_ny; iy++)
01750
             if (obscount[ix][iy] > 0) {
01751
01752
                /* Write output... */
01753
                fprintf(out, "\n");
01754
01755
                /* Loop over altitudes... */
01756
                for (iz = 0; iz < ctl->prof_nz; iz++) {
01757
01758
                  /* 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);
01760
01761
01762
01763
                  /* Get meteorological data... */
                   press = P(z);
01764
01765
                   intpol_met_time(met0, met1, t, press, lon, lat,
01766
                                      NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
01767
                  /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
 * cos(lat * M_PI / 180.);
01768
01769
01770
01772
                  mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
01773
                  01774
01775
                            tmean[ix][iy] / obscount[ix][iy],
01776
                            z, lon, lat, press, temp, mmr, h2o, o3, obsmean[ix][iy] / obscount[ix][iy]);
01777
01778
01779
01780
01781
         /* Close file... */
01782
         if (t == ctl->t_stop)
01783
01784
           fclose(out);
01785 }
```

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 1789 of file libtrac.c.

```
01793
                    {
01794
01795
        static FILE *out;
01797
        static double rmax2, t0, t1, x0[3], x1[3];
01798
01799
        static int init, ip, iq;
01800
01801
        /* Init... */
01802
        if (!init) {
01803
          init = 1;
01804
01805
          /* Write info... */
          printf("Write station data: %s\n", filename);
01806
01807
01808
           /* Create new file... */
01809
          if (!(out = fopen(filename, "w")))
01810
            ERRMSG("Cannot create file!");
01811
           /* Write header... */
01812
01813
          fprintf(out,
                   "# $1 = time [s]\n"
"# $2 = altitude [km]\n"
01814
01815
01816
                   "# $3 = longitude [deg] \n" "# <math>$4 = latitude [deg] \n");
           01817
01818
01819
01820
          fprintf(out, "\n");
01821
01822
           /* Set geolocation and search radius... */
01823
           geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
01824
           rmax2 = gsl_pow_2(ctl->stat_r);
01825
01826
01827
         /* Set time interval for output... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01828
01829
01830
        /* Loop over air parcels... */
for (ip = 0; ip < atm->np; ip++) {
01831
01832
01833
01834
01835
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
01836
             continue;
01837
          /* Check station flag... */
if (ctl->qnt_stat >= 0)
01838
01839
01840
            if (atm->q[ctl->qnt_stat][ip])
01841
01842
01843
           /* Get Cartesian coordinates... */
          geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
01844
01845
01846
           /\star Check horizontal distance... \star/
```

5.14 libtrac.h 117

```
01847
          if (DIST2(x0, x1) > rmax2)
01848
           continue;
01849
          /* Set station flag... */
if (ctl->qnt_stat >= 0)
atm->q[ctl->qnt_stat][ip] = 1;
01850
01851
01852
01853
01854
           /* Write data...
          01855
01856
          for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01857
01858
            fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01859
01860
01861
          fprintf(out, "\n");
01862
01863
        /* Close file... */
if (t == ctl->t_stop)
01864
01865
01866
          fclose(out);
01867 }
```

Here is the call graph for this function:



5.14 libtrac.h

```
00001 /*
00002
        This file is part of MPTRAC.
00003
00004
        \ensuremath{\mathsf{MPTRAC}} is free software: you can redistribute it and/or modify
00005
        it under the terms of the GNU General Public License as published by
00006
        the Free Software Foundation, either version 3 of the License, or
00007
        (at your option) any later version.
00008
00009
        \ensuremath{\mathsf{MPTRAC}} is distributed in the hope that it will be useful,
        but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
00012
        GNU General Public License for more details.
00013
00014
        You should have received a copy of the GNU General Public License
00015
        along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>.
00016
00017
        Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00034 #include <ctype.h>
00035 #include <gsl/gsl_const_mksa.h>
00036 #include <gsl/gsl_math.h>
00037 #include <gsl/gsl_randist.h>
00038 #include <gsl/gsl_rng.h>
00039 #include <gsl/gsl_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
00054 #define ALLOC(ptr, type, n)
```

```
if((ptr=calloc((size_t)(n), sizeof(type)))==NULL)
00056
         ERRMSG("Out of memory!");
00057
00059 #define DIST(a, b) sqrt(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) {
00070 printf("\nError (%s, %s, 1%d): %s\n\n", 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
00086 #define NORM(a) sgrt(DOTP(a, a))
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)
00111
00113 #define PRINT_TIMER(id) timer(#id, id, 3)
00114
00115 /* -----
00116
         Constants...
00117
00118
00120 #define G0 9.80665
00121
00123 #define H0 7.0
00124
00126 #define P0 1013.25
00127
00129 #define RE 6367.421
00130
00131 /* -----
         Dimensions...
00132
00133
00134
00136 #define LEN 5000
00137
00139 #define NP 10000000
00140
00142 #define NO 10
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 /* -----
         Structs...
00169
00170
```

5.14 libtrac.h 119

```
00171
00173 typedef struct {
00174
00176
        int nq;
00177
00179
        char qnt_name[NQ][LEN];
00180
00182
        char qnt_unit[NQ][LEN];
00183
00185
        char qnt_format[NQ][LEN];
00186
00188
        int qnt_m;
00189
00191
        int qnt_rho;
00192
00194
        int qnt_r;
00195
00197
        int qnt_ps;
00198
00200
        int qnt_p;
00201
00203
        int qnt_t;
00204
00206
        int qnt_u;
00207
00209
        int qnt_v;
00210
00212
        int qnt_w;
00213
00215
        int qnt_h2o;
00216
        int qnt_o3;
00219
00221
        int qnt_theta;
00222
00224
        int qnt_pv;
00225
        int qnt_tice;
00228
00230
        int qnt_tnat;
00231
        int qnt_stat;
00234
00236
        int direction;
00237
00239
        double t_start;
00240
00242
        double t_stop;
00243
00245
        double dt mod:
00246
00248
        double dt_met;
00249
00251
        int met_np;
00252
00254
        double met_p[EP];
00255
00258
        int isosurf;
00259
00261
        char balloon[LEN];
00262
00264
        double turb_dx_trop;
00265
00267
        double turb_dx_strat;
00268
00270
        double turb_dz_trop;
00271
00273
        double turb_dz_strat;
00274
        double turb_meso;
00277
00279
        double tdec_trop;
00280
00282
        double tdec_strat;
00283
        char atm_basename[LEN];
00286
00288
        char atm_gpfile[LEN];
00289
00291
        double atm_dt_out;
00292
00294
        int atm_filter;
00295
00297
        char csi_basename[LEN];
00298
00300
        double csi_dt_out;
00301
```

```
00303
        char csi_obsfile[LEN];
00304
00306
        double csi_obsmin;
00307
00309
        double csi modmin;
00310
00312
        int csi_nz;
00313
00315
        double csi_z0;
00316
        double csi_z1;
00318
00319
00321
        int csi_nx;
00322
00324
        double csi_lon0;
00325
        double csi_lon1;
00327
00328
00330
        int csi_ny;
00331
00333
        double csi_lat0;
00334
        double csi_lat1;
00336
00337
00339
        char grid_basename[LEN];
00340
00342
        char grid_gpfile[LEN];
00343
00345
        double grid_dt_out;
00346
00348
        int grid_sparse;
00349
00351
        int grid_nz;
00352
00354
        double grid_z0;
00355
00357
        double grid_z1;
00358
00360
        int grid_nx;
00361
00363
        double grid_lon0;
00364
        double grid_lon1;
00366
00367
00369
        int grid_ny;
00370
00372
        double grid_lat0;
00373
        double grid_lat1;
00375
00376
        char prof_basename[LEN];
00379
00381
        char prof_obsfile[LEN];
00382
        int prof_nz;
00384
00385
        double prof_z0;
00388
00390
        double prof_z1;
00391
        int prof_nx;
00393
00394
00396
        double prof_lon0;
00397
00399
        double prof_lon1;
00400
00402
        int prof_ny;
00403
00405
        double prof_lat0;
00406
00408
        double prof_lat1;
00409
        char stat_basename[LEN];
00411
00412
00414
        double stat_lon;
00415
00417
        double stat_lat;
00418
00420
        double stat_r;
00421
00422 } ctl_t;
00423
00425 typedef struct {
00426
00428
        int np;
00429
00431
        double time[NP];
```

5.14 libtrac.h 121

```
00432
00434
        double p[NP];
00435
00437
        double lon[NP];
00438
00440
        double lat[NP]:
00441
00443
        double q[NQ][NP];
00444
00446
        double up[NP];
00447
00449
       double vp[NP];
00450
00452
        double wp[NP];
00453
00454 } atm_t;
00455
00457 typedef struct {
00458
00460
        double time;
00461
00463
        int nx;
00464
00466
        int ny;
00467
00469
        int np;
00470
00472
        double lon[EX];
00473
00475
        double lat[EY];
00476
        double p[EP];
00479
00481
        double ps[EX][EY];
00482
        float pl[EX][EY][EP];
00484
00485
        float t[EX][EY][EP];
00488
00490
        float u[EX][EY][EP];
00491
        float v[EX][EY][EP];
00493
00494
00496
        float w[EX][EY][EP];
00497
00499
        float h2o[EX][EY][EP];
00500
        float o3[EX][EY][EP];
00502
00503
00504 } met_t;
00505
00506 /* -----
00507
         Functions...
00508
00509
00511 void cart2geo(
       double *x,
00513
        double *z,
00514
        double *lon,
00515
        double *lat);
00516
00518 double deg2dx(
00519
        double dlon,
00520
        double lat);
00521
00523 double deg2dy(
00524
        double dlat);
00525
00527 double dp2dz(
00528
       double dp,
00529
        double p);
00530
00532 double dx2deg(
00533
        double dx,
        double lat);
00534
00535
00537 double dy2deg(
00538
        double dy);
00539
00541 double dz2dp(
00542
       double dz,
00543
        double p);
00544
00546 void geo2cart(
00547
        double z,
00548
        double lon,
00549
        double lat.
```

```
double *x);
00551
00553 void get_met(
        ctl_t * ctl,
char *metbase,
00554
00555
00556
        double t,
        met_t * met0,
met_t * met1);
00557
00558
00559
00561 void get_met_help(
00562
        double t.
00563
        int direct.
00564
        char *metbase,
00565
        double dt_met,
00566
        char *filename);
00567
00569 void intpol_met_2d(
00570
        double array[EX][EY],
00571
        int ix,
00572
        int iy,
00573
        double wx,
00574
        double wy,
00575
        double *var);
00576
00578 void intpol_met_3d(
00579
        float array[EX][EY][EP],
00580
        int ip,
00581
        int ix,
00582
        int iy,
        double wp, double wx,
00583
00584
00585
        double wy,
00586
        double *var);
00587
00593
        double lat,
00594
        double *ps,
00595
        double *t,
00596
        double *u,
00597
        double *v.
00598
        double *w,
00599
        double *h2o,
00600
        double *o3);
00601
00603 void intpol_met_time(
00604
        met_t * met0,
met_t * met1,
00605
00606
        double ts,
        double p, double lon,
00607
00608
00609
        double lat,
00610
        double *ps,
00611
        double *t,
00612
        double *u,
00613
        double *v,
00614
        double *w,
00615
        double *h2o,
00616
        double *o3);
00617
00619 void jsec2time(
00620
       double jsec,
00621
        int *year,
00622
        int *mon,
00623
        int *day,
00624
        int *hour,
00625
        int *min,
00626
        int *sec,
00627
        double *remain);
00628
00630 int locate(
        double *xx,
00631
00632
        int n,
00633
        double x);
00634
00636 void read_atm(
        const char *filename,
ctl_t * ctl,
atm_t * atm);
00637
00638
00639
00640
00642 void read_ctl(
00643
        const char *filename,
00644
        int argc,
00645
        char *argv[],
ctl_t * ctl);
00646
```

5.14 libtrac.h 123

```
00647
00649 void read_met(
00650
       ctl_t * ctl,
        char *filename,
00651
00652
        met_t * met);
00653
00655 void read_met_extrapolate(
00656
       met_t * met);
00657
00659 void read_met_help(
00660 int ncid,
00661
        char *varname.
00662
        char *varname2,
00663
        met_t * met,
00664
        float dest[EX][EY][EP],
00665
        float scl);
00666
00668 void read_met_ml2pl(
00669 ctl_t * ctl,
00670 met_t * met,
00671
        float var[EX][EY][EP]);
00672
00674 void read_met_periodic(
00675
       met_t * met);
00676
00678 double scan_ctl(
00679
        const char *filename,
00680
        int argc,
        char *argv[],
const char *varname,
00681
00682
00683
       int arridx,
const char *defvalue,
00684
00685
        char *value);
00686
00688 void time2jsec(
00689
        int year,
00690
        int mon,
00691
        int day,
00692
        int hour,
00693
        int min,
00694
        int sec,
        double remain,
00695
00696
        double *jsec);
00697
00699 void timer(
00700
       const char *name,
00701
        int id,
00702
        int mode);
00703
00704 /* Get tropopause pressure... */
00705 double tropopause(
00706 double t,
00707
        double lat);
00708
00710 void write_atm(
00711
        const char *filename,
        ctl_t * ctl,
atm_t * atm,
00712
00713
00714
        double t);
00715
00717 void write_csi(
00718 const char *filename,
        ctl_t * ctl,
atm_t * atm,
00719
00720
00721
        double t);
00722
00724 void write_grid(
00725
        const char *filename,
        ctl_t * ctl,
00726
00727
        met_t * met0,
        met_t * met1,
atm_t * atm,
00728
00729
00730
        double t);
00731
00733 void write_prof(
00734
       const char *filename,
00735
        ctl_t * ctl,
00736
        met_t * met0,
        met_t * met1,
atm_t * atm,
00737
00738
00739
        double t);
00740
00742 void write_station(
00743
        const char *filename,
        ctl_t * ctl,
atm_t * atm,
00744
00745
00746
        double t);
```

5.15 match.c File Reference

Calculate deviations between two trajectories.

Functions

• int main (int argc, char *argv[])

5.15.1 Detailed Description

Calculate deviations between two trajectories.

Definition in file match.c.

5.15.2 Function Documentation

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

Definition at line 28 of file match.c.

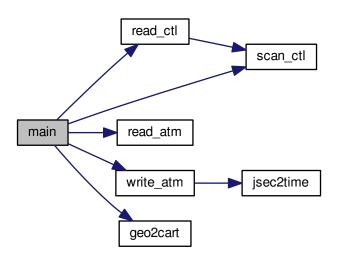
```
00030
                          {
00031
00032
         ctl_t ctl;
00033
00034
         atm_t *atm1, *atm2, *atm3;
00035
00036
        FILE *out;
00037
00038
        char filename[LEN];
00039
00040
        double filter_dt, x1[3], x2[3], dh, dq[NQ], dv, 1h = 0, 1t = 0, 1v = 0;
00041
00042
       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

```
"# \$%d = trajectory time [s]\n"
00079
                   "# \$%d = vertical length of trajectory [km]\n"
                   "# \$%d = horizontal length of trajectory [km]\n"
00080
                   "# \$%d = vertical deviation [km]\n"
00081
                   "# \$%d = horizontal deviation [km]\n",
00082
00083
                   5 + ctl.nq, 6 + ctl.nq, 7 + ctl.nq, 8 + ctl.nq, 9 + ctl.nq);
         for (iq = 0; iq < ctl.nq; iq+)
    fprintf(out, "# $%d = %s deviation [%s]\n", ctl.nq + iq + 10,</pre>
00085
         ctl.qnt_name[iq], ctl.qnt_unit[iq]);
fprintf(out, "\n");
00086
00087
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... */
for (ip1 = 0; ip1 < atm2->np; ip1++) {
00096
00097
             n = 0;
00098
              atm2->p[ip1] = 0;
00099
              for (iq = 0; iq < ctl.nq; iq++)</pre>
              atm2->q[iq][ip1] = 0;
for (ip2 = 0; ip2 < atm2->np; ip2++)
00100
00101
00102
                if (fabs(atm2->time[ip1] - atm2->time[ip2]) < filter_dt) {</pre>
                  atm2->p[ip1] += atm3->p[ip2];
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++)
00109
00110
                atm2->q[iq][ip1] /= n;
00111
00112
           /* Write filtered data... */
sprintf(filename, "%s.filt", argv[3]);
00113
00114
00115
           write_atm(filename, &ctl, atm2, 0);
00116
00117
00118
         /* Loop over air parcels (reference data)... */
         for (ip2 = 0; ip2 < atm2->np; ip2++) {
00119
00120
00121
            /* Get trajectory length... */
           if (ip2 > 0) {
00122
00123
              geo2cart(0, atm2->lon[ip2 - 1], atm2->lat[ip2 - 1], x1);
00124
              geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00125
              1h += DIST(x1, x2);
              lv += fabs(Z(atm2->p[ip2 - 1]) - Z(atm2->p[ip2]));
lt = fabs(atm2->time[ip2] - atm2->time[0]);
00126
00127
00128
00129
00130
           /* Init... */
00131
           n = 0;

dh = 0;
00132
           dv = 0;
00133
           for (iq = 0; iq < ctl.nq; iq++)</pre>
             dq[iq] = 0;
00135
00136
            geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00137
00138
            /* Find corresponding time step (test data)... */
           for (ip1 = 0; ip1 < atml->np; ip1++)
  if (fabs(atml->time[ip1] - atm2->time[ip2])
  < (filter ? filter_dt : 0.1)) {</pre>
00139
00140
00141
00142
00143
                /* Calculate deviations... */
00144
                geo2cart(0, atm1->lon[ip1], atm1->lat[ip1], x1);
                dh += DIST(x1, x2);
dv += Z(atm1->p[ip1]) - Z(atm2->p[ip2]);
00145
00146
                for (iq = 0; iq < ctl.nq; iq++)</pre>
00147
00148
                   dq[iq] += atm1->q[iq][ip1] - atm2->q[iq][ip2];
00149
                n++;
00150
             }
00151
            /* Write output... */
00152
           if (n > 0) {
00153
00154
             fprintf(out, "%.2f %.4f %.4f %.4f",
00155
                     atm2->time[ip2], Z(atm2->p[ip2]),
                       atm2->lon[ip2], atm2->lat[ip2]);
00156
              for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl.qnt_format[iq], atm2->q[iq][ip2]);
00157
00158
00159
00160
              fprintf(out, " %.2f %g %g %g %g", lt, lv, lh, dv / n, dh / n);
00161
              for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl.qnt_format[iq], dq[iq] / n);</pre>
00162
00163
00164
```

```
00165
00166
            fprintf(out, "\n");
00167
       }
00168
00169
00170
        /* Close file... */
00171
        fclose(out);
00172
00173
        /* Free... */
00174
        free(atm1);
00175
        free (atm2);
00176
       free (atm3);
00177
00178
        return EXIT_SUCCESS;
00179 }
```

Here is the call graph for this function:



5.16 match.c

```
00001 /*
00002
          This file is part of MPTRAC.
00003
          MPTRAC is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by
00004
00005
          the Free Software Foundation, either version 3 of the License, or
00006
00007
          (at your option) any later version.
80000
00009
          \ensuremath{\mathsf{MPTRAC}} is distributed in the hope that it will be useful,
          but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
00012
          GNU General Public License for more details.
00013
00014
          You should have received a copy of the GNU General Public License
00015
          along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/>.
00016
00017
         Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00025 #include "libtrac.h"
00026 #include <gsl/gsl_sort.h>
00027
00028 int main(
00029 int argc,
00030 char *argv[]) {
00031
```

5.16 match.c 127

```
00032
        ctl_t ctl;
00033
00034
        atm_t *atm1, *atm2, *atm3;
00035
00036
        FILE *out:
00037
         char filename[LEN];
00039
00040
         double filter_dt, x1[3], x2[3], dh, dq[NQ], dv, 1h = 0, 1t = 0, 1v = 0;
00041
00042
         int filter, ip1, ip2, iq, n;
00043
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)
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);
filter_dt = scan_ctl(argv[1], argc, argv, "FILTER_DT", -1, "0", NULL);
00054
00055
00056
00057
00058
         /* Read atmospheric data... */
00059
         read_atm(argv[2], &ctl, atml);
00060
        read_atm(argv[3], &ctl, atm2);
00061
00062
         /* Write info... */
00063
         printf("Write transport deviations: %s\n", argv[4]);
00064
         /\star Create output file... \star/
00065
         if (!(out = fopen(argv[4], "w")))
    ERRMSG("Cannot create file!");
00066
00067
00068
00069
         /* Write header... */
00070
         fprintf(out,
00071
                  "# $1 = time [s] \n"
                  "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00072
00073
         for (iq = 0; iq < ctl.nq; iq++)
fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00074
00075
00076
                     ctl.qnt_unit[iq]);
00077
         fprintf(out,
00078
                  "# \$%d = trajectory time [s]\n"
                   "# \$%d = vertical length of trajectory [km]\n"
00079
                   "# \$%d = horizontal length of trajectory [km]\n"
08000
                  "# $%d = vertical deviation [km]\n"
00081
00082
                  "# $%d = horizontal deviation [km]\n",
00083
                  5 + ctl.nq, 6 + ctl.nq, 7 + ctl.nq, 8 + ctl.nq, 9 + ctl.nq);
         for (iq = 0; iq < ctl.nq; iq++)
  fprintf(out, "# $%d = %s deviation [%s]\n", ctl.nq + iq + 10,</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;
              atm2->p[ip1] = 0;

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

atm2->q[iq][ip1] = 0;

for (ip2 = 0; ip2 < atm2->np; ip2++)
00098
00099
00100
00101
00102
                if (fabs(atm2->time[ip1] - atm2->time[ip2]) < filter_dt) {</pre>
00103
                  atm2->p[ip1] += atm3->p[ip2];
00104
                  for (iq = 0; iq < ctl.nq; iq++)</pre>
                    atm2->q[iq][ip1] += atm3->q[iq][ip2];
00105
00106
                  n++;
00107
00108
              atm2->p[ip1] /= n;
00109
              for (iq = 0; iq < ctl.nq; iq++)</pre>
00110
                atm2->q[iq][ip1] /= n;
00111
00112
00113
            /* Write filtered data... */
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++) {
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
               1h += 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++)</pre>
00134
              dq[iq] = 0;
00135
             geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00136
00137
00138
             /\star Find corresponding time step (test data)... \star/
            for (ip1 = 0; ip1 < atm1->np; ip1++)
   if (fabs(atm1->time[ip1] - atm2->time[ip2])
00139
00140
                     < (filter ? filter_dt : 0.1)) {
00141
00142
00143
                  /* Calculate deviations... */
                  geo2cart(0, atm1->lon[ip1], atm1->lat[ip1], x1);
00144
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
             /\star Write output... \star/
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, " ");
  fprintf(out, ctl.qnt_format[iq], atm2->q[iq][ip2]);
00157
00158
00159
00160
               fprintf(out, " %.2f %g %g %g %g", lt, lv, lh, dv / n, dh / n);
for (iq = 0; iq < ctl.nq; iq++) {
   fprintf(out, " ");
   fprintf(out, ctl.qnt_format[iq], dq[iq] / n);</pre>
00161
00162
00163
00164
00165
                fprintf(out, "\n");
00166
00167
            }
00168
00169
00170
          /* Close file... */
00171
          fclose(out);
00172
00173
          /* Free... */
00174
          free(atm1);
          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

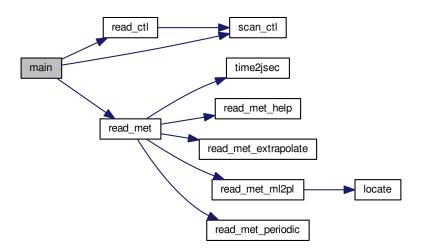
5.17.2.1 int main (int argc, char * argv[])

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... */
        read_ctl(argv[1], argc, argv, &ctl);
z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00050
00051
00052
00053
         /* Loop over files... *
00054
        for (i = 3; i < argc; i++) {
00055
00056
           /* Read meteorological data... */
           if (!(in = fopen(argv[i], "r")))
00057
00058
            continue;
00059
          else
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++)
00074
            for (iy = 0; iy < met->ny; iy++) {
00075
               timem[ix][iy] += met->time;
               tm[ix][iy] += met->t[ix][iy][ip];
um[ix][iy] += met->u[ix][iy][ip];
00076
00077
00078
               vm[ix][iy] += met->v[ix][iy][ip];
00079
               wm[ix][iy] += met->w[ix][iy][ip];
00080
               h2om[ix][iy] += met->h2o[ix][iy][ip];
               o3m[ix][iy] += met->o3[ix][iy][ip];
psm[ix][iy] += met->ps[ix][iy];
00081
00082
00083
               np[ix][iy]++;
00084
00085
        }
00086
00087
        /* Create output file... */
        printf("Write meteorological data file: %s\n", argv[2]);
if (!(out = fopen(argv[2], "w")))
00088
00089
          ERRMSG("Cannot create file!");
00090
00091
         /* Write header... */
00092
00093
        fprintf(out,
00094
                  "# $1
                         = time [s]\n"
                 "# $2 = altitude [km]\n"
00095
                  "# $3 = longitude [deg]\n"
00096
                 "# $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
00102
                 "# $9 = vertical wind [hPa/s]\n"
00103
                 "# $10 = H20 volume mixing ratio [1]\n"
00104
                 "# $11 = 03 volume mixing ratio [1]\n"
```

```
"# $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++)
    if (met->lon[ix] >= 180)
      fprintf(out, "%.2f %g %g,",
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],
vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00113
00114
00115
00116
00117
00118
                                      psm[ix][iy] / np[ix][iy]);
00119
                 for (ix = 0; ix < met\rightarrownx; ix++)
                   00120
00121
00122
00123
                                     met=>lol(ix), met=>let(iy), met=>p(iy),
tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
psm[ix][iy] / np[ix][iy]);
00124
00125
00126
00127
00128
00129
00130
              /* Close file... */
00131
             fclose(out);
00132
             /* Free... */
00133
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
       MPTRAC is free software: you can redistribute it and/or modify
00005
        it under the terms of the GNU General Public License as published by
00006
       the Free Software Foundation, either version 3 of the License, or
00007
        (at your option) any later version.
80000
00009
       MPTRAC is distributed in the hope that it will be useful,
00010
       but WITHOUT ANY WARRANTY; without even the implied warranty of
```

5.18 met map.c 131

```
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
         GNU General Public License for more details.
00013
00014
        You should have received a copy of the GNU General Public License
00015
        along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>.
00016
00017
         Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028
        int argc,
00029
        char *argv[]) {
00030
00031
        ctl_t ctl;
00032
00033
        met t *met;
00034
00035
        FILE *in, *out;
00036
00037
         static double dz, dzmin = 1e10, z, timem[EX][EY], psm[EX][EY], tm[EX][EY],
00038
          um[EX][EY], vm[EX][EY], wm[EX][EY], h2om[EX][EY], o3m[EX][EY];
00039
00040
        static int i, ip, ip2, ix, iy, np[EX][EY];
00041
00042
         /* Allocate... */
00043
        ALLOC(met, met_t, 1);
00044
00045
         /* Check arguments... */
00046
        if (argc < 4)
00047
           ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00048
00049
        /\star Read control parameters... \star/
        read_ctl(argv[1], argc, argv, &ctl);
z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00050
00051
00052
         /* Loop over files... */
00054
         for (i = 3; i < argc; i++) {</pre>
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
           /* Find nearest pressure level... */
for (ip2 = 0; ip2 < met->np; ip2++) {
   dz = fabs(Z(met->p[ip2]) - z);
00063
00064
00065
00066
              if (dz < dzmin) {
00067
                dzmin = dz;
00068
                ip = ip2;
00069
             }
00070
00071
           /* Average data... */
00073
           for (ix = 0; ix < met->nx; ix++)
00074
             for (iy = 0; iy < met->ny; iy++) {
00075
               timem[ix][iy] += met->time;
               tm[ix][iy] += met->t[ix][iy][ip];
um[ix][iy] += met->u[ix][iy][ip];
00076
00077
00078
                vm[ix][iy] += met->v[ix][iy][ip];
00079
                wm[ix][iy] += met->w[ix][iy][ip];
00080
                h2om[ix][iy] += met->h2o[ix][iy][ip];
                o3m[ix][iy] += met->o3[ix][iy][ip];
psm[ix][iy] += met->ps[ix][iy];
00081
00082
00083
                np[ix][iy]++;
00084
00085
        }
00086
00087
        /* Create output file... */
        printf("Write meteorological data file: %s\n", argv[2]);
if (!(out = fopen(argv[2], "w")))
00088
00089
           ERRMSG("Cannot create file!");
00090
00091
         /* Write header... */
00092
00093
        fprintf(out,
00094
                  "# $1
                         = time [s]\n"
                  "# $2 = altitude [km]\n"
00095
                  "# $3 = longitude [deg] \n"
00096
00097
                  "# $4 = latitude [deg] \n"
                         = pressure [hPa]\n"
00098
                  "# $5
00099
                  "# $6
                         = temperature [K]\n"
                  "# $7 = zonal wind [m/s] n"
00100
                  "# $8 = meridional wind [m/s]\n"
00101
00102
                  "# $9 = vertical wind [hPa/s]\n"
```

```
"# $10 = H20 volume mixing ratio [1]\n"
                   "# $11 = 03 volume mixing ratio [1]\n'
"# $12 = surface pressure [hPa]\n");
00104
00105
00106
00107
         /* Write data... */
         for (iy = 0; iy < met->ny; iy++) {
    fprintf(out, "\n");
00108
00109
00110
            for (ix = 0; ix < met->nx; ix++)
             00111
00112
00113
00114
00115
00116
00117
00118
                          psm[ix][iy] / np[ix][iy]);
            00119
00120
00121
                          timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
met->lon[ix], met->lat[iy], met->p[ip],
00122
00123
                          met=>loft[ix], met=>lat([y], met=>p[x]);
tm([ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
psm[ix][iy] / np[ix][iy]);
00124
00125
00126
00127
00128
00129
00130
         /* Close file... */
00131
         fclose(out);
00132
00133
         /* Free... */
00134
         free (met);
00135
00136
         return EXIT_SUCCESS;
00137 }
```

5.19 met prof.c File Reference

Extract vertical profile from meteorological data.

Functions

• int main (int argc, char *argv[])

5.19.1 Detailed Description

Extract vertical profile from meteorological data.

Definition in file met prof.c.

5.19.2 Function Documentation

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

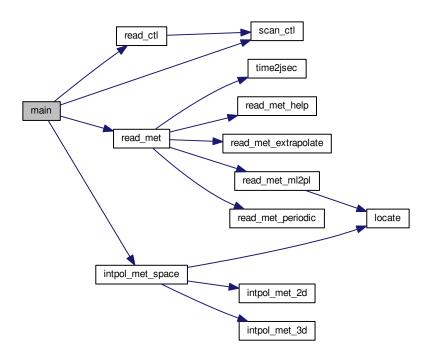
Definition at line 38 of file met_prof.c.

```
00041
00042
            ctl_t ctl;
00043
00044
           met t *met;
00045
           FILE *in, *out;
00047
           static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ],
w, wm[NZ], h2o, h2om[NZ], o3, o3m[NZ], ps, psm[NZ];
00048
00049
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> <prof.tab> <met0> [ <met1> ... ]");
00060
00061
            /* Read control parameters... */
           /* Read control parameters... */
read_ctl(argv[1], argc, argv, &ctl);
z0 = scan_ctl(argv[1], argc, argv, "Z0", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "Z1", -1, "60", NULL);
dz = scan_ctl(argv[1], argc, argv, "DZ", -1, "1", NULL);
lon0 = scan_ctl(argv[1], argc, argv, "LON0", -1, "0", NULL);
dlon = scan_ctl(argv[1], argc, argv, "LON1", -1, "0", NULL);
dlon = scan_ctl(argv[1], argc, argv, "DLON", -1, "1", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "LAT0", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "LAT1", -1, "0", NULL);
dlat = scan_ctl(argv[1], argc, argv, "DLAT", -1, "1", NULL);
00062
00063
00064
00066
00067
00068
00069
00070
00071
00072
00073
            /\star Loop over input files... \star/
00074
            for (i = 3; i < argc; i++) {</pre>
00075
00076
              /* Read meteorological data... */
if (!(in = fopen(argv[i], "r")))
00078
                  continue;
00079
00080
                 fclose(in);
               read_met(&ctl, argv[i], met);
00081
00082
00083
               /* Average... */
               for (z = z0; z \le z1; z += dz) {
00085
                  iz = (int) ((z - z0) / dz);
00086
                   if (iz < 0 || iz > NZ)
                     ERRMSG("Too many altitudes!");
00087
                   for (lon = lon0; lon <= lon1; lon += dlon)</pre>
00088
                     for (lat = lat0; lat <= lat1; lat += dlat) {
00089
                        intpol_met_space(met, P(z), lon, lat, &ps,
00090
00091
                                                  &t, &u, &v, &w, &h2o, &o3);
                         if (gsl_finite(t) && gsl_finite(u)
00092
00093
                              && gsl_finite(v) && gsl_finite(w)) {
                            timem[iz] += met->time;
lonm[iz] += lon;
latm[iz] += lat;
00094
00095
00097
                            tm[iz] += t;
00098
                            um[iz] += u;
00099
                            vm[iz] += v;
                            wm[iz] += w;
00100
                            h2om[iz] += h2o;
00101
                            o3m[iz] += o3;
psm[iz] += ps;
00102
00103
                            np[iz]++;
00104
00105
00106
                     }
              }
00107
00108
00110
             /* Normalize... */
            for (z = z0; z <= z1; z += dz) {
  iz = (int) ((z - z0) / dz);
  if (np[iz] > 0) {
    timem[iz] /= np[iz];
}
00111
00112
00113
00114
00115
                  lonm[iz] /= np[iz];
00116
                  latm[iz] /= np[iz];
                  tm[iz] /= np[iz];
um[iz] /= np[iz];
vm[iz] /= np[iz];
wm[iz] /= np[iz];
00117
00118
00119
00120
                  h2om[iz] /= np[iz];
00122
                  o3m[iz] /= np[iz];
00123
                  psm[iz] /= np[iz];
00124
               } else {
                  timem[iz] = GSL_NAN;
00125
00126
                  lonm[iz] = GSL_NAN;
```

```
00127
              latm[iz] = GSL_NAN;
              tm[iz] = GSL_NAN;
um[iz] = GSL_NAN;
00128
00129
              vm[iz] = GSL_NAN;
wm[iz] = GSL_NAN;
00130
00131
              h2om[iz] = GSL_NAN;
o3m[iz] = GSL_NAN;
psm[iz] = GSL_NAN;
00132
00133
00134
00135
00136
00137
         /* Create output file... */
00138
         printf("Write meteorological data file: %s\n", argv[2]);
00139
00140
         if (!(out = fopen(argv[2], "w")))
00141
           ERRMSG("Cannot create file!");
00142
         /* Write ...
fprintf(out,
    "# $1
         /* Write header... */
00143
00144
                           = time [s]\n"
00146
                   "# $2 = altitude [km]\n"
00147
                   "# $3 = longitude [deg] \n"
                   "# $4 = latitude [deg]\n"
00148
                   "# $5 = pressure [hPa]\n"
"# $6 = temperature [K]\n"
"# $7 = zonal wind [m/s]\n"
"# $8 = meridional wind [m/s]\n"
00149
00150
00151
00152
00153
                   "# $9 = vertical wind [hPa/s] n"
                   "# \$10 = H20 volume mixing ratio [1]\n"
"# \$11 = 03 volume mixing ratio [1]\n"
"# \$12 = surface pressure [hPa]\n\n");
00154
00155
00156
00157
         00158
00159
00160
00161
00162
00163
00164
00165
00166
         /* Close file... */
00167
         fclose(out);
00168
         /* Free... */
00169
00170
         free (met);
00171
00172
         return EXIT_SUCCESS;
00173 }
```

5.20 met prof.c 135

Here is the call graph for this function:



5.20 met_prof.c

```
00001 /*
00002
         This file is part of MPTRAC.
00003
00004
         MPTRAC is free software: you can redistribute it and/or modify
00005
         it under the terms of the GNU General Public License as published by
         the Free Software Foundation, either version 3 of the License, or
00006
00007
         (at your option) any later version.
80000
00009
         \ensuremath{\mathsf{MPTRAC}} is distributed in the hope that it will be useful,
         but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
00012
         GNU General Public License for more details.
00013
         You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
00016
00017
         Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00025 #include "libtrac.h"
00026
00027 /* -----
00028
          Dimensions...
00029
00030
00031 /\star Maximum number of altitudes. \star/
00032 #define NZ 1000 \,
00033
00034 /* --
00035
          Main...
00036
00037
00038 int main(
00039
         int argc,
00040
         char *argv[]) {
00041
00042
         ctl_t ctl;
00043
```

```
00044
          met_t *met;
00045
00046
           FILE *in, *out;
00047
           static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ], lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ], w, wm[NZ], h2o, h2om[NZ], o3, o3m[NZ], ps, psm[NZ];
00048
00049
00051
00052
           static int i, iz, np[NZ];
00053
00054
            /* Allocate... */
00055
           ALLOC(met, met_t, 1);
00056
00057
            /* Check arguments... */
00058
           if (argc < 4)
00059
              ERRMSG("Give parameters: <ctl>   <met0> [ <met1> ... ]");
00060
00061
           /* Read control parameters... */
           read_ctl(argv[1], argc, argv, &ctl);
00062
          read_ctl(argv[1], argc, argv, &ctl);
z0 = scan_ctl(argv[1], argc, argv, "Z0", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "Z1", -1, "60", NULL);
dz = scan_ctl(argv[1], argc, argv, "DZ", -1, "1", NULL);
lon0 = scan_ctl(argv[1], argc, argv, "LONO", -1, "0", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "LONO", -1, "0", NULL);
dlon = scan_ctl(argv[1], argc, argv, "DLON", -1, "1", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "LATO", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "LAT1", -1, "0", NULL);
dlat = scan_ctl(argv[1], argc, argv, "DLAT", -1, "1", NULL);
00063
00064
00065
00066
00067
00068
00069
00070
00071
00072
00073
           /* Loop over input files... */
00074
           for (i = 3; i < argc; i++) {
00075
00076
               /* Read meteorological data... */
00077
              if (!(in = fopen(argv[i], "r")))
00078
                 continue;
00079
              else
08000
                fclose(in);
              read_met(&ctl, argv[i], met);
00082
00083
               /* Average... */
              for (z = z0; z <= z1; z += dz) {
  iz = (int) ((z - z0) / dz);
00084
00085
                 <u>if</u> (iz < 0 || iz > NZ)
00086
00087
                    ERRMSG("Too many altitudes!");
                  for (lon = lon0; lon <= lon1; lon += dlon)</pre>
00088
00089
                    for (lat = lat0; lat <= lat1; lat += dlat) {</pre>
00090
                       intpol_met_space(met, P(z), lon, lat, &ps,
                       &t, &u, &v, &w, &h2o, &o3);
if (gsl_finite(t) && gsl_finite(u)
00091
00092
                            && gsl_finite(v) && gsl_finite(w)) {
00093
00094
                          timem[iz] += met->time;
00095
                          lonm[iz] += lon;
                          latm[iz] += lat;
00096
00097
                          tm[iz] += t;
                          um[iz] += u;
00098
00099
                          vm[iz] += v;
                          wm[iz] += w;
00100
00101
                          h2om[iz] += h2o;
00102
                          o3m[iz] += o3;
00103
                          psm[iz] += ps;
00104
                          np[iz]++;
00105
00106
                    }
00107
             }
00108
00109
00110
           /* Normalize... */
           for (z = z0; z <= z1; z += dz) {
  iz = (int) ((z - z0) / dz);
00111
00112
              if (np[iz] > 0) {
00113
00114
                timem[iz] /= np[iz];
                 lonm[iz] /= np[iz];
00115
                 latm[iz] /= np[iz];
00116
                 tm[iz] /= np[iz];
um[iz] /= np[iz];
vm[iz] /= np[iz];
00117
00118
00119
00120
                 wm[iz] /= np[iz];
00121
                 h2om[iz] /= np[iz];
                o3m[iz] /= np[iz];
psm[iz] /= np[iz];
00122
00123
00124
              } else {
00125
                 timem[iz] = GSL_NAN;
                 lonm[iz] = GSL_NAN;
latm[iz] = GSL_NAN;
00126
00127
                tm[iz] = GSL_NAN;
um[iz] = GSL_NAN;
00128
00129
                 vm[iz] = GSL_NAN;
00130
```

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

5.21 met_sample.c File Reference

Sample meteorological data at given geolocations.

Functions

• int main (int argc, char *argv[])

5.21.1 Detailed Description

Sample meteorological data at given geolocations.

Definition in file met_sample.c.

5.21.2 Function Documentation

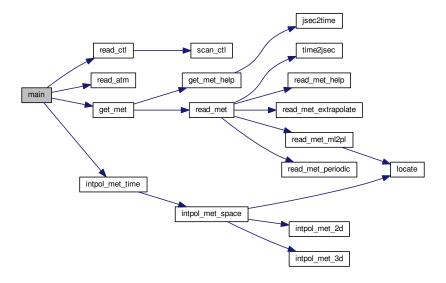
5.21.2.1 int main (int argc, char * argv[])

Definition at line 31 of file met_sample.c.

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

5.22 met sample.c 139

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.

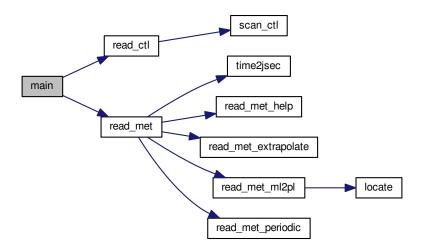
```
00029
00030
00031
                ctl_t ctl;
00032
00033
               met t *met;
00034
                FILE *in, *out;
00036
00037
                static double timem[EP][EY], psm[EP][EY], tm[EP][EY], um[EP][EY],
                   vm(EP][EY], vhm[EP][EY], wm[EP][EY], h2om[EP][EY], o3m[EP][EY],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

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

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

5.24 met zm.c 143

Here is the call graph for this function:



5.24 met_zm.c

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

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

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

5.25 smago.c File Reference

Estimate horizontal diffusivity based on Smagorinsky theory.

Functions

• int main (int argc, char *argv[])

5.25.1 Detailed Description

Estimate horizontal diffusivity based on Smagorinsky theory.

Definition in file smago.c.

5.25.2 Function Documentation

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

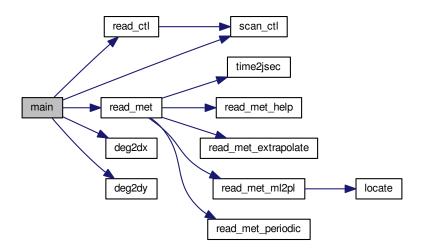
Definition at line 8 of file smago.c.

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

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

5.26 smago.c 147

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
         /* Read control parameters... */
read_ctl(argv[1], argc, argv, &ctl);
z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00029
00030
00031
00032
00033
          /* 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

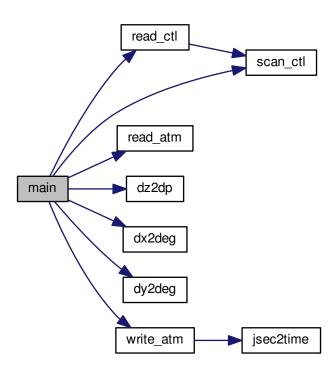
5.27.2.1 int main (int argc, char * argv[])

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 151

5.28 split.c

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

```
00090
           else
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 153

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 155

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 }
```

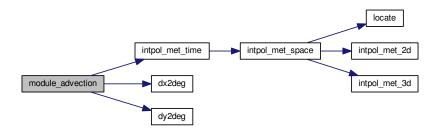
5.31.2.2 void module_advection (met_t * met0, met_t * met1, atm_t * atm, int ip, double dt)

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:



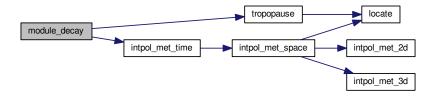
5.31.2.3 void module_decay (ctl_t * ctl, met_t * met0, met_t * met1, atm_t * atm, int ip, double dt)

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.

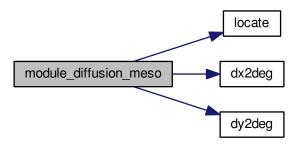
Definition at line 497 of file trac.c.

```
{
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];
00529
00530
            v[2] = met0 -> v[ix][iy + 1][iz];
           v[3] = met0->v[ix + 1][iy + 1][iz];
v[4] = met0->v[ix][iy][iz + 1];
00531
00532
           v[5] = met0->v[ix + 1][iy][iz + 1];
v[6] = met0->v[ix][iy + 1][iz + 1];
00533
00534
00535
            v[7] = met0 -> v[ix + 1][iy + 1][iz + 1];
00536
00537
            w[0] = met0->w[ix][iy][iz];
00538
            w[1] = met0->w[ix + 1][iy][iz];
           w[2] = met0->w[ix][iy + 1][iz];
00539
           w[3] = met0->w[ix + 1][iy + 1][iz];
w[4] = met0->w[ix][iy][iz + 1];
00540
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:



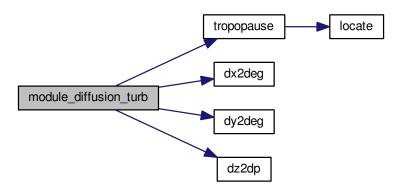
5.31.2.5 void module diffusion turb (ctl t * ctl, atm t * atm, int ip, double dt, gsl rng * rng)

Calculate turbulent diffusion.

Definition at line 608 of file trac.c.

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

Here is the call graph for this function:



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

Force air parcels to stay on isosurface.

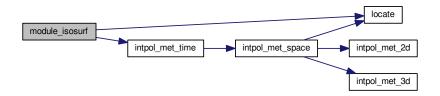
Definition at line 653 of file trac.c.

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

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

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

Here is the call graph for this function:



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

Interpolate meteorological data for air parcel positions.

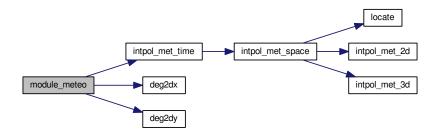
Definition at line 768 of file trac.c.

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

```
0 ? atm->lon[ip] - 1. : atm->lon[ip] + 1.),
00827
                                   atm->lat[ip], NULL, NULL, NULL, &v1, NULL, NULL, NULL);
00828
               vort += (v1 - v) / 1000.
                / ((atm->lon[ip] >= 0 ? -1 : 1) * deg2dx(1., atm->lat[ip]));
00829
00830
            intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00831
       lon[ip],
00832
                                 (atm->lat[ip] >=
            0 ? atm->lat[ip] - 1. : atm->lat[ip] + 1.), NULL, NULL, 
&u1, NULL, NULL, NULL, NULL); 
vort += (u1 - u) / 1000. / ((atm->lat[ip] >= 0 ? -1 : 1) * deg2dy(1.));
00833
00834
00835
00836
00837
            /* Potential temperature gradient... */
00838
            p1 = 0.85 * atm -> p[ip];
00839
            intpol_met_time(met0, met1, atm->time[ip], p1, atm->lon[ip],
            atm->lat[ip], NULL, &t1, NULL, NULL, NULL, NULL, NULL, NULL); grad = (t1 * pow(P0 / p1, 0.286) - t * pow(P0 / atm->p[ip], 0.286))
00840
00841
               / (100. * (pl - atm->p[ip]));
00842
00843
00844
             /* Calculate PV... */
            atm \rightarrow q[ctl \rightarrow qnt_pv][ip] = -1e6 * G0 * vort * grad;
00845
00846
00847
00848
          /* Calculate T_ice (Marti and Mauersberger, 1993)... */
if (ctl->qnt_tice >= 0)
00849
          atm->q[ctl->qnt\_tice][ip] = -2663.5
00850
00851
               / (log10(4e-6 * atm->p[ip] * 100.) - 12.537);
00852
00853
          /\star Calculate T_NAT (Hanson and Mauersberger, 1988)... \star/
         if (ctl->qnt_tnat >= 0) {
  p_hno3 = 9e-9 * atm->p[ip] / 1.333224;
  p_h2o = 4e-6 * atm->p[ip] / 1.333224;
00854
00855
00856
00857
            term1 = 38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o);
            term2 = 0.009179 - 0.00088 * log10(p_h2o);
00858
            b = term1 / term2;
c = -11397.0 / term2;
00859
00860
            x1 = (-b + \text{sqrt}(b * b - 4. * c)) / 2.;

x2 = (-b - \text{sqrt}(b * b - 4. * c)) / 2.;
00861
00862
            if (x1 > 0)
00863
00864
              atm->q[ctl->qnt_tnat][ip] = x1;
00865
            if (x2 > 0)
               atm->q[ctl->qnt\_tnat][ip] = x2;
00866
00867
00868 }
```

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)

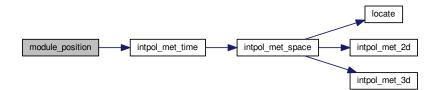
Check position of air parcels.

Definition at line 872 of file trac.c.

```
00876
00877
00878 double ps;
00879
```

```
/* Calculate modulo... */
         atm->lon[ip] = fmod(atm->lon[ip], 360);
atm->lat[ip] = fmod(atm->lat[ip], 360);
00882
00883
         /* Check latitude... */
while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
00884
00885
           if (atm->lat[ip] > 90) {
   atm->lat[ip] = 180 - atm->lat[ip];
00886
00887
00888
              atm->lon[ip] += 180;
00889
00890
           if (atm->lat[ip] < -90) {
             atm->lat[ip] = -180 - atm->lat[ip];
00891
              atm->lon[ip] += 180;
00892
00893
00894
00895
         /* Check longitude... */
while (atm->lon[ip] < -180)
atm->lon[ip] += 360;
00896
00897
00898
00899
         while (atm->lon[ip] >= 180)
00900
           atm->lon[ip] -= 360;
00901
         00902
00903
00904
00905
                            NULL, NULL, NULL, NULL, NULL);
00906
00907
          /* Check pressure... */
         if (atm->p[ip] > ps)
00908
         atm->p[ip] = ps;
else if (atm->p[ip] < met0->p[met0->np - 1])
atm->p[ip] = met0->p[met0->np - 1];
00909
00910
00911
00912 }
```

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)

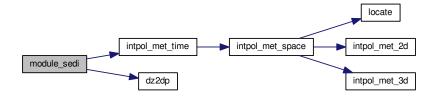
Calculate sedimentation of air parcels.

Definition at line 916 of file trac.c.

```
00922
00923
00924
        /* Coefficients for Cunningham slip-flow correction (Kasten, 1968): */
00925
        const double A = 1.249, B = 0.42, C = 0.87;
00926
00927
         /\star Specific gas constant for dry air [J/(kg K)]: \star/
00928
        const double R = 287.058;
00929
        /* Average mass of an air molecule [kg/molec]: \star/ const double m = 4.8096e-26;
00930
00931
00932
00933
        double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
00934
00935
         /* Check if parameters are available... */
00936
        if (ctl->qnt_r < 0 || ctl->qnt_rho < 0)</pre>
00937
          return:
00938
00939
        /* Convert units... */
```

```
p = 100 * atm->p[ip];
00941
         r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
00942
         rho_p = atm->q[ctl->qnt_rho][ip];
00943
00944
         /* Get temperature... */
         intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00945
      lon[ip],
00946
                           atm->lat[ip], NULL, &T, NULL, NULL, NULL, NULL, NULL);
00947
00948
         /\star Density of dry air... \star/
         rho = p / (R * T);
00949
00950
        /* Dynamic viscosity of air... */
eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
00951
00952
00953
        /* Thermal velocity of an air molecule... */ v = sqrt(8 * GSL_CONST_MKSA_BOLTZMANN * T / (M_PI * m));
00954
00955
00956
00957
         /* Mean free path of an air molecule... */
00958
         lambda = 2 * eta / (rho * v);
00959
00960
         /* Knudsen number for air... */
00961
        K = lambda / r_p;
00962
00963
         /* Cunningham slip-flow correction... */
00964
         G = 1 + K * (A + B * exp(-C / K));
00965
00966
         /* Sedimentation (fall) velocity... */
00967
00968
           2. * gsl_pow_2(r_p) * (rho_p -
                                     rho) * GSL_CONST_MKSA_GRAV_ACCEL / (9. * eta) * G;
00969
00970
        /* Calculate pressure change... */ atm->p[ip] += dz2dp(v_p * dt / 1000., atm->p[ip]);
00971
00972
00973 }
```

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)

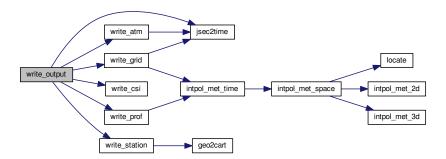
Write simulation output.

Definition at line 977 of file trac.c.

```
00983
              {
00984
00985
      char filename[LEN];
00986
00987
      double r;
00988
00989
      int year, mon, day, hour, min, sec;
00990
00991
      /* Get time... */
00992
      jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
00993
      00994
00995
00996
00997
00998
        write_atm(filename, ctl, atm, t);
```

```
00999
01000
                                     01001
01002
01003
01004
01005
01006
01007
01008
                                     /* Write CSI data... */
                                     if (ctl->csi_basename[0] != '-') {
01009
                                     sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
write_csi(filename, ctl, atm, t);
01010
01011
01012
01013
                                     /* Write profile data... */
if (ctl->prof_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
    write prof/filename, ctl = the ctl = t
01014
01015
01016
                                              write_prof(filename, ctl, met0, met1, atm, t);
01017
01018
01019
                                 /* Write station data... */
if (ctl->stat_basename[0] != '-') {
   sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
   write_station(filename, ctl, atm, t);
01020
01021
01022
01023
01024
01025 }
```

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
       /* Initialize MPI... */
00181
00182
       MPI_Init(&argc, &argv);
00183
       MPI_Comm_rank (MPI_COMM_WORLD, &rank);
00184
       MPI_Comm_size(MPI_COMM_WORLD, &size);
```

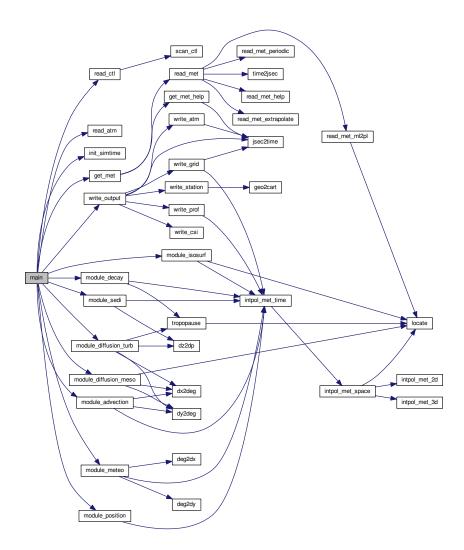
```
00185 #endif
00186
00187
         /* Check arguments... */
        if (argc < 5)
00188
00189
          ERRMSG("Give parameters: <dirlist> <ctl> <atm in> <metbase>");
00190
        /* Open directory list... */
if (!(dirlist = fopen(argv[1], "r")))
00191
00192
00193
         ERRMSG("Cannot open directory list!");
00194
        /* Loop over directories... */
while (fscanf(dirlist, "%s", dirname) != EOF) {
00195
00196
00197
00198
           /* MPI parallelization... */
00199
          if ((++ntask) % size != rank)
00200
            continue;
00201
00202
00203
             Initialize model run...
00204
00205
00206
           /* Set timers... */
          START_TIMER (TIMER_TOTAL);
00207
00208
          START TIMER (TIMER INIT);
00209
00210
           /* Allocate... */
00211
           ALLOC(atm, atm_t, 1);
00212
           ALLOC(met0, met_t, 1);
00213
           ALLOC(met1, met_t, 1);
00214
          ALLOC(dt, double,
00215
                NP);
00216
00217
           /* Read control parameters... */
00218
           sprintf(filename, "%s/%s", dirname, argv[2]);
00219
           read_ctl(filename, argc, argv, &ctl);
00220
00221
           /* Initialize random number generators... */
           gsl_rng_env_setup();
00223
           for (i = 0; i < NTHREADS; i++)</pre>
00224
            rng[i] = gsl_rng_alloc(gsl_rng_default);
00225
          /* Read atmospheric data... */
sprintf(filename, "%s/%s", dirname, argv[3]);
read_atm(filename, &ctl, atm);
00226
00227
00228
00229
00230
           /* Get simulation time interval... */
00231
           init_simtime(&ctl, atm);
00232
00233
           /* Get rounded start time... */
           if (ctl.direction == 1)
00234
            t0 = floor(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00235
00236
00237
            t0 = ceil(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00238
00239
           /* Set timers... */
00240
          STOP TIMER (TIMER INIT);
00241
00242
00243
              Loop over timesteps...
00244
00245
          /* Loop over timesteps... */
for (t = t0; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;</pre>
00246
00247
00248
                t += ctl.direction * ctl.dt_mod) {
00249
00250
             /\star Adjust length of final time step... \star/
            if (ctl.direction * (t - ctl.t_stop) > 0)
00251
00252
              t = ctl.t_stop;
00253
00254
             /* Set time steps for air parcels... */
00255
             for (ip = 0; ip < atm\rightarrownp; ip++)
00256
               if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
                    00257
00258
00259
                 dt[ip] = t - atm->time[ip];
00260
               else
00261
                 dt[ip] = GSL_NAN;
00262
00263
             /\star Get meteorological data... \star/
             START_TIMER(TIMER_INPUT);
00264
00265
             get_met(&ctl, argv[4], t, met0, met1);
00266
             STOP_TIMER(TIMER_INPUT);
00267
             /* Initialize isosurface... */
00268
00269
             {\tt START\_TIMER} (TIMER_ISOSURF);
00270
             if (t == t0)
00271
              module_isosurf(&ctl, met0, met1, atm, -1);
```

```
STOP_TIMER(TIMER_ISOSURF);
00273
00274
             /* Advection... */
00275
            START_TIMER (TIMER_ADVECT);
00276 #pragma omp parallel for default(shared) private(ip) 00277 for (ip = 0; ip < atm->np; ip++)
             if (gsl_finite(dt[ip]))
00279
                 module_advection(met0, met1, atm, ip, dt[ip]);
00280
            STOP_TIMER(TIMER_ADVECT);
00281
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()]);
            STOP_TIMER(TIMER_DIFFTURB);
00289
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)
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... */
            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... */
            START_TIMER(TIMER_POSITION);
00316
00317 #pragma omp parallel for default(shared) private(ip)
00318
           for (ip = 0; ip < atm->np; ip++)
              module_position(met0, met1, atm, ip);
00319
00320
            STOP_TIMER(TIMER_POSITION);
00321
00322
             /* Meteorological data... */
00323
            START_TIMER (TIMER_METEO);
00324 \#pragma omp parallel for default(shared) private(ip)
           for (ip = 0; ip < atm->np; ip++)
    module_meteo(&ctl, met0, met1, atm, ip);
00325
00326
00327
            STOP TIMER (TIMER METEO);
00329
00330
            START_TIMER (TIMER_DECAY);
{\tt 00331~\#pragma~omp~parallel~for~default(shared)~private(ip)}
            for (ip = 0; ip < atm->np; ip++)
00332
             if (gsl_finite(dt[ip]))
  module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00333
00334
            STOP_TIMER(TIMER_DECAY);
00335
00336
00337
             /* Write output...
00338
            START_TIMER(TIMER_OUTPUT);
             write_output(dirname, &ctl, met0, met1, atm, t);
00339
00340
            STOP_TIMER(TIMER_OUTPUT);
00341
00342
00343
00344
             Finalize model run...
00345
00346
00347
           /* Report timers...
00348
          STOP_TIMER(TIMER_TOTAL);
00349
          PRINT_TIMER (TIMER_TOTAL);
00350
          PRINT_TIMER(TIMER_INIT);
          PRINT_TIMER(TIMER_INPUT);
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);
00358
          PRINT_TIMER (TIMER_METEO);
```

```
00359
          PRINT_TIMER(TIMER_POSITION);
00360
          PRINT_TIMER(TIMER_SEDI);
00361
         00362
00363
00364
00365
00366
         00367
00368
00369
00370
00371
         /* Report problem size... */
printf("SIZE_NP = %d\n", atm->np);
printf("SIZE_TASKS = %d\n", size);
printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00372
00373
00374
00375
00376
          /* 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
00389 /* Finalize MPI... */
00390 MPI_Finalize();
00391 #endif
00392
00393
       return EXIT_SUCCESS;
00394 }
```

5.32 trac.c 169

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.
80000
00009
          \ensuremath{\mathsf{MPTRAC}} is distributed in the hope that it will be useful,
          but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
00012
          GNU General Public License for more details.
00013
          You should have received a copy of the GNU General Public License \,
00014
00015
         along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>.
00016
00017
         Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 #ifdef MPI
00028 #include "mpi.h"
00029 #endif
```

```
00030
00031 /* -----
00032
        Defines...
00033
00034
00036 #define TIMER_TOTAL 0
00039 #define TIMER_INIT 1
00040
00042 #define TIMER INPUT 2
00043
00045 #define TIMER OUTPUT 3
00046
00048 #define TIMER_ADVECT 4
00049
00051 #define TIMER_DECAY 5
00052
00054 #define TIMER_DIFFMESO 6
00055
00057 #define TIMER_DIFFTURB 7
00058
00060 #define TIMER_ISOSURF 8
00061
00063 #define TIMER_METEO 9
00064
00066 #define TIMER_POSITION 10
00067
00069 #define TIMER_SEDI 11
00070
00071 /* -----
00072
       Functions...
00073
00074
00076 void init_simtime(
00077 ctl_t * ctl,
00078 atm_t * atm);
00079
00081 void module_advection(
00082
       met_t * met0,
00083
        met_t * met1,
00084
       atm_t * atm,
00085
       int ip,
00086
       double dt);
00087
00089 void module_decay(
00090
       ctl_t * ctl,
00091
       met_t * met0,
00092
       met_t * met1,
00093
       atm_t * atm,
00094
       int ip,
00095
        double dt);
00096
00098 void module_diffusion_meso(
       ctl_t * ctl,

met_t * met0,

met_t * met1,

atm_t * atm,
00099
00100
00101
00102
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,
00119
        met_t * met1,
        atm_t * atm,
00120
00121
        int ip);
00122
00124 void module_meteo(
00125
       ctl_t * ctl,
00126
        met_t * met0,
       met_t * met1,
atm_t * atm,
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);
```

5.32 trac.c 171

```
00137
00139 void module_sedi(
        ctl_t * ctl,
met_t * met0,
00140
00141
        met_t * met1,
atm_t * atm,
00142
00143
        int ip,
00144
00145
        double dt);
00146
00148 void write_output(
00149
        const char *dirname,
00150
        ctl_t * ctl,
        met_t * met0,
00151
00152
        met_t * met1,
00153
        atm_t * atm,
00154
        double t);
00155
00156 /*
00157
        Main...
00158
00159
00160 int main(
00161
        int argc,
        char *argv[]) {
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
        char dirname[LEN], filename[LEN];
00174
00175
00176
        double *dt, t, t0;
00177
00178
        int i, ip, ntask = 0, rank = 0, size = 1;
00179
00180 #ifdef MPI
00181
        /* Initialize MPT... */
00182
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00184
        MPI_Comm_size(MPI_COMM_WORLD, &size);
00185 #endif
00186
         /* Check arguments... */
00187
00188
        if (argc < 5)
          ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00189
00190
00191
        /* Open directory list... */
00192
        if (!(dirlist = fopen(argv[1], "r")))
          ERRMSG("Cannot open directory list!");
00193
00194
00195
        /* Loop over directories... */
while (fscanf(dirlist, "%s", dirname) != EOF) {
00196
00197
00198
           /* MPI parallelization... */
          if ((++ntask) % size != rank)
00199
00200
            continue;
00201
00202
00203
             Initialize model run...
00204
00205
00206
           /* Set timers... */
           START_TIMER(TIMER_TOTAL);
00207
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]);
read_ctl(filename, argc, argv, &ctl);
00217
00218
00219
00220
00221
           /* Initialize random number generators... */
00222
           gsl_rng_env_setup();
00223
           for (i = 0; i < NTHREADS; i++)</pre>
00224
            rng[i] = gsl_rng_alloc(gsl_rng_default);
00225
```

```
00226
          /* Read atmospheric data... */
          sprintf(filename, "%s/%s", dirname, argv[3]);
00227
00228
          read_atm(filename, &ctl, atm);
00229
          /* Get simulation time interval... */
00230
00231
          init simtime(&ctl, atm);
00232
00233
          /* Get rounded start time... */
00234
          if (ctl.direction == 1)
           t0 = floor(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
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
00246
          /* Loop over timesteps... */
00247
          for (t = t0; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;</pre>
               t += ctl.direction * ctl.dt_mod) {
00248
00249
00250
            /* Adjust length of final time step... */
            if (ctl.direction * (t - ctl.t_stop) > 0)
00251
00252
              t = ctl.t_stop;
00253
00254
            /\star Set time steps for air parcels... \star/
            00255
00256
00257
00258
00259
                dt[ip] = t - atm->time[ip];
00260
               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
            START_TIMER(TIMER_ISOSURF);
00269
00270
            if (t == t0)
00271
              module_isosurf(&ctl, met0, met1, atm, -1);
00272
            STOP_TIMER(TIMER_ISOSURF);
00273
00274
            /* Advection... */
00275
            START_TIMER(TIMER_ADVECT);
00276 #pragma omp parallel for default(shared) private(ip)
00277
            for (ip = 0; ip < atm->np; ip++)
00278
             if (gsl_finite(dt[ip]))
00279
                module_advection(met0, met1, atm, ip, dt[ip]);
00280
            STOP_TIMER(TIMER_ADVECT);
00281
            /* Turbulent diffusion...
            START_TIMER(TIMER_DIFFTURB);
00283
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
            STOP_TIMER(TIMER_DIFFTURB);
00289
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
00300
            /* Sedimentation...
            START_TIMER(TIMER_SEDI);
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]);
            STOP_TIMER(TIMER_SEDI);
00306
00307
00308
            /* 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
```

5.32 trac.c 173

```
STOP_TIMER(TIMER_ISOSURF);
00314
00315
            /* Position... */
00316
            START_TIMER(TIMER_POSITION);
00317 #pragma omp parallel for default(shared) private(ip)
           for (ip = 0; ip < atm->np; ip++)
   module_position(met0, met1, atm, ip);
00318
00319
00320
            STOP_TIMER(TIMER_POSITION);
00321
00322
            /* Meteorological data... */
            START_TIMER (TIMER_METEO);
00323
00324 \#pragma omp parallel for default(shared) private(ip)
00325
           for (ip = 0; ip < atm->np; ip++)
00326
              module_meteo(&ctl, met0, met1, atm, ip);
00327
            STOP_TIMER(TIMER_METEO);
00328
00329
            /* Decay... */
            START_TIMER(TIMER_DECAY);
00330
00331 #pragma omp parallel for default(shared) private(ip)
           for (ip = 0; ip < atm->np; ip++)
00332
00333
             if (gsl_finite(dt[ip]))
00334
                module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00335
            STOP_TIMER(TIMER_DECAY);
00336
00337
            /* Write output...
            START_TIMER (TIMER_OUTPUT);
00338
00339
            write_output(dirname, &ctl, met0, met1, atm, t);
00340
            STOP_TIMER(TIMER_OUTPUT);
00341
00342
00343
00344
             Finalize model run...
00345
00346
00347
          /* Report timers..
          STOP_TIMER(TIMER_TOTAL);
00348
          PRINT_TIMER (TIMER_TOTAL);
00349
          PRINT_TIMER (TIMER_INIT);
00350
00351
          PRINT_TIMER (TIMER_INPUT);
00352
          PRINT_TIMER (TIMER_OUTPUT);
00353
          PRINT_TIMER(TIMER_ADVECT);
          PRINT_TIMER(TIMER_DECAY);
PRINT_TIMER(TIMER_DIFFMESO);
00354
00355
00356
          PRINT_TIMER (TIMER_DIFFTURB);
          PRINT_TIMER (TIMER_ISOSURF);
00357
00358
          PRINT_TIMER(TIMER_METEO);
00359
          PRINT_TIMER (TIMER_POSITION);
00360
          PRINT_TIMER (TIMER_SEDI);
00361
00362
          /* Report memory usage... */
          printf("MEMORY_ATM = %g MByte\n", 2. * sizeof(atm_t) / 1024. / 1024.);
printf("MEMORY_METEO = %g MByte\n", 2. * sizeof(met_t) / 1024. / 1024.);
00363
00364
          printf("MEMORY_DYNAMIC = %g MByte\n",
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
          /* Free random number generators... */
00377
00378
          for (i = 0; i < NTHREADS; i++)
00379
           gsl_rng_free(rng[i]);
00380
          /* Free... */
00381
00382
          free(atm);
00383
          free (met 0);
00384
          free (met1);
00385
          free(dt);
00386
00387
00388 #ifdef MPI
00389
       /* Finalize MPI... */
00390
       MPI_Finalize();
00391 #endif
00392
00393
        return EXIT SUCCESS;
00394 }
00395
00397
00398 void init_simtime(
00399
       ctl t * ctl.
```

```
00400
       atm_t * atm) {
00401
00402
        /* Set inital and final time... */
        if (ctl->direction == 1) {
   if (ctl->t_start < -1e99)</pre>
00403
00404
            ctl->t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00405
          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>
00410
           ctl->t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00411
          if (ctl->t start < -1e99)
00412
           ctl->t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00413
00414
        /* Check time... */
00415
        if (ctl->direction * (ctl->t_stop - ctl->t_start) <= 0)</pre>
00416
          ERRMSG("Nothing to do!");
00417
00418 }
00419
00421
00422 void module_advection(
00423
       met_t * met0,
met_t * met1,
00424
        atm_t * atm,
00425
        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
        /* Get position of the mid point... */  xm[0] = atm->lon[ip] + dx2deg(0.5 * dt * v[0] / 1000., atm->lat[ip]); \\ xm[1] = atm->lat[ip] + dy2deg(0.5 * dt * v[1] / 1000.); 
00436
00437
00438
00439
        xm[2] = atm - p[ip] + 0.5 * dt * v[2];
00440
00441
        /* Interpolate meteorological data for mid point... */
        00442
00443
00444
                        &v[0], &v[1], &v[2], NULL, NULL);
00445
00446
        /* Save new position... */
00447
        atm->time[ip] += dt;
        atm->lon[ip] += dx2deg(dt * v[0] / 1000., xm[1]);
atm->lat[ip] += dy2deg(dt * v[1] / 1000.);
00448
00449
00450
        atm->p[ip] += dt * v[2];
00451 }
00452
00454
00455 void module decay(
00456
       ctl_t * ctl,
met_t * met0,
00457
00458
        met_t * met1,
00459
        atm_t * atm,
00460
        int ip,
00461
       double dt) {
00462
00463
        double ps, pt, tdec;
00464
        /\star Check lifetime values... \star/
00465
        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
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, 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... */
00485
          if (atm->p[ip] \le pt)
```

5.32 trac.c 175

```
tdec = ctl->tdec_strat;
00487
00488
            tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
      p[ip]);
00489
00490
         /* Calculate exponential decay... */
00492
         atm \rightarrow q[ctl \rightarrow qnt_m][ip] *= exp(-dt / tdec);
00493 }
00494
00496
00497 void module_diffusion_meso(
00498
        ctl_t * ctl,
00499
         met_t * met0,
        met_t * met1,
atm_t * atm,
00500
00501
00502
         int ip,
00503
        double dt,
00504
        gsl_rng * rng) {
00505
00506
        double r, rs, u[16], v[16], w[16], usig, vsig, wsig;
00507
00508
        int ix, iy, iz;
00509
00510
        /\star Calculate mesoscale velocity fluctuations... \star/
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]);
00515
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][iy + 1][iz];
u[3] = met0->u[ix + 1][iy + 1][iz];
00521
00523
           u[4] = met0 -> u[ix][iy][iz + 1];
00524
           u[5] = met0 -> u[ix + 1][iy][iz + 1];
00525
           u[6] = met0 -> u[ix][iy + 1][iz + 1];
           u[7] = met0 -> u[ix + 1][iy + 1][iz + 1];
00526
00527
00528
           v[0] = met0 -> v[ix][iy][iz];
           v[1] = met0 -> v[ix + 1][iy][iz];
00529
00530
           v[2] = met0 -> v[ix][iy + 1][iz];
00531
           v[3] = met0 -> v[ix + 1][iy + 1][iz];
           v[4] = met0 -> v[ix][iy][iz + 1];
00532
           v[5] = met0->v[ix + 1][iy][iz + 1];
v[6] = met0->v[ix][iy + 1][iz + 1];
00533
00534
           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];
w[2] = met0->w[ix][iy + 1][iz];
00538
00539
           w[3] = met0->w[ix + 1][iy + 1][iz];
w[4] = met0->w[ix][iy][iz + 1];
00540
00541
00542
           w[5] = met0 -> w[ix + 1][iy][iz + 1];
00543
           w[6] = met0 -> w[ix][iy + 1][iz + 1];
00544
           w[7] = met0 -> w[ix + 1][iy + 1][iz + 1];
00545
00546
           /* Get indices... */
00547
           ix = locate(met1->lon, met1->nx, atm->lon[ip]);
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
           u[10] = met1->u[ix][iy + 1][iz];
00555
           u[11] = met1->u[ix + 1][iy + 1][iz];
           u[12] = met1->u[ix][iy][iz + 1];
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];
00559
00560
00561
           v[8] = met1->v[ix][iy][iz];
00562
           v[9] = met1 -> v[ix + 1][iy][iz];
00563
           v[10] = met1->v[ix][iy + 1][iz];
           v[11] = met1->v[ix, [iy, 1], [iz],
v[11] = met1->v[ix + 1][iy + 1][iz];
v[12] = met1->v[ix][iy][iz + 1];
00564
00565
00566
           v[13] = met1 -> v[ix + 1][iy][iz + 1];
00567
           v[14] = met1 -> v[ix][iy + 1][iz + 1];
           v[15] = met1 -> v[ix + 1][iy + 1][iz + 1];
00568
00569
           w[8] = met1->w[ix][iy][iz];
w[9] = met1->w[ix + 1][iy][iz];
00570
00571
```

```
w[10] = met1->w[ix][iy + 1][iz];
         w[11] = met1->w[ix + 1][iy + 1][iz];
w[12] = met1->w[ix][iy][iz + 1];
00573
00574
          w[13] = met1->w[ix + 1][iy][iz + 1];
00575
          w[14] = met1->w[ix][iy + 1][iz + 1];
00576
          w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00577
00578
00579
          /* Get standard deviations of local wind data... */
00580
          usig = gsl_stats_sd(u, 1, 16);
00581
          vsig = gsl_stats_sd(v, 1, 16);
00582
          wsig = gsl_stats_sd(w, 1, 16);
00583
00584
          /* Set temporal correlations for mesoscale fluctuations... */
00585
          r = 1 - 2 * fabs(dt) / ctl->dt_met;
00586
          rs = sqrt(1 - r * r);
00587
00588
          /* Calculate mesoscale wind fluctuations... */
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 * qsl_ran_qaussian_ziqqurat(rnq,
00597
                                                             ctl->turb_meso * wsig);
00598
         /* Calculate air parcel displacement... */
atm->lon[ip] += dx2deg(atm->up[ip] * dt / 1000., atm->lat[ip]);
atm->lat[ip] += dy2deg(atm->vp[ip] * dt / 1000.);
00599
00600
00601
00602
          atm->p[ip] += atm->wp[ip] * dt;
00603
00604 }
00605
00607
00608 void module diffusion turb(
       ctl_t * ctl,
        atm_t * atm,
00610
00611
        int ip,
00612
       double dt,
00613
       gsl_rng * rng) {
00614
00615
       double dx, dz, pt, p0, p1, w;
00616
00617
        /* Get tropopause pressure... */
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
         w = 1;
00625
        else if (atm->p[ip] < p1)
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 \rightarrow turb_dx_trop + (1 - w) * ctl \rightarrow 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) {
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)))
                      / 1000.);
00641
00642
00643
00644
        /* Vertical turbulent diffusion... */
00645
        if (dz > 0)
00646
         atm->p[ip]
00647
           += dz2dp(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dz * fabs(dt)))
00648
                     / 1000., atm->p[ip]);
00649 }
00650
00652
00653 void module_isosurf(
00654
      ctl_t * ctl,
00655
       met_t * met0,
       met_t * met1,
atm_t * atm,
00656
00657
       int ip) {
00658
```

5.32 trac.c 177

```
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
00668
        /\star Check control parameter... \star/
        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... */
          if (ctl->isosurf == 1)
00685
            for (ip2 = 0; ip2 < atm->np; ip2++)
00686
              iso[ip2] = atm->p[ip2];
00687
00688
          /* Save density... */
00689
          else if (ctl->isosurf == 2)
00690
            for (ip2 = 0; ip2 < atm->np; ip2++) {
00691
              intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
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
          /* Save potential temperature... */
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
00701
                               atm->lon[ip2], atm->lat[ip2], NULL, &t, NULL, NULL,
                               NULL, NULL, NULL);
00702
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: %s\n", ctl->balloon);
00711
00712
            /* Open file... */
            if (!(in = fopen(ctl->balloon, "r")))
00713
              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)
00719
                  ERRMSG("Too many data points!");
00720
00721
00722
            /* Check number of points... */
00723
              ERRMSG("Could not read any data!");
00724
00725
00726
            /* Close file... */
00727
            fclose(in);
00728
00729
00730
          /* Leave initialization... */
00731
00732
00733
00734
        /★ Restore pressure... ★/
00735
        if (ctl->isosurf == 1)
00736
         atm->p[ip] = iso[ip];
00737
00738
       /* Restore density... */
else if (ctl->isosurf == 2) {
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->p[ip] = iso[ip] * t;
00743
00744
```

```
00745
       /* Restore potential temperature... */
       else if (ctl->isosurf == 3) {
00746
00747
         intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
     lon[ip],
          00748
00749
00750
00751
00752
        /* Interpolate pressure... */
00753
        else if (ctl->isosurf == 4) {
         if (atm->time[ip] <= ts[0])</pre>
00754
          atm->p[ip] = ps[0];
else if (atm->time[ip] >= ts[n - 1])
00755
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,
00772
        atm_t * atm,
00773
        int ip) {
00774
       double b, c, ps, p1, p_hno3, p_h2o, t, t1, term1, term2,
    u, u1, v, v1, w, x1, x2, h2o, o3, grad, vort;
00775
00776
00777
        /* Interpolate meteorological data... */
00778
        intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00779
     lon[ip],
00780
                         atm->lat[ip], &ps, &t, &u, &v, &w, &h2o, &o3);
00781
00782
        /* Set surface pressure... */
00783
        if (ctl->qnt_ps >= 0)
00784
         atm->q[ctl->qnt_ps][ip] = ps;
00785
00786
        /* Set pressure... */
00787
       if (ctl->qnt_p >= 0)
00788
          atm->q[ctl->qnt_p][ip] = atm->p[ip];
00789
       /* Set temperature... */
if (ctl->qnt_t >= 0)
00790
00791
         atm->q[ctl->qnt_t][ip] = t;
00792
00793
00794
        /\star Set zonal wind... \star/
00795
        if (ctl->qnt_u >= 0)
00796
         atm->q[ctl->qnt_u][ip] = u;
00797
00798
       /* Set meridional wind... */
00799
       if (ctl->qnt_v >= 0)
00800
          atm->q[ctl->qnt_v][ip] = v;
00801
00802
        /* Set vertical velocity... */
       if (ctl->qnt_w >= 0)
00803
         atm->q[ctl->qnt_w][ip] = w;
00804
00805
00806
        /* Set water vapor vmr... */
00807
        if (ct1->qnt_h2o>=0)
00808
         atm->q[ctl->qnt_h2o][ip] = h2o;
00809
00810
        /* Set ozone vmr... */
       if (ctl->qnt_o3 >= 0)
00811
          atm->q[ctl->qnt_o3][ip] = o3;
00812
00813
00814
        /\star Calculate potential temperature... \star/
        if (ctl->qnt_theta >= 0)
  atm->q[ctl->qnt_theta][ip] = t * pow(P0 / atm->p[ip], 0.286);
00815
00816
00817
00818
        /* Calculate potential vorticity... */
00819
        if (ctl->qnt_pv >= 0) {
00820
          /* Absolute vorticity... */
vort = 2 * 7.2921e-5 * sin(atm->lat[ip] * M_PI / 180.);
if (fabs(atm->lat[ip]) < 89.) {
00821
00822
00823
            intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00824
00825
                             (atm->lon[ip] >=
00826
                              0 ? atm->lon[ip] - 1. : atm->lon[ip] + 1.),
00827
                             atm->lat[ip], NULL, NULL, NULL, &v1, NULL, NULL, NULL);
            vort += (v1 - v) / 1000.
  / ((atm->lon[ip] >= 0 ? -1 : 1) * deg2dx(1., atm->lat[ip]));
00828
00829
```

5.32 trac.c 179

```
intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00832
                           (atm->lat[ip] >=
          0 ? atm->lat[ip] - 1. : atm->lat[ip] + 1.), NULL, NULL, &u1, NULL, NULL, NULL, NULL); vort += (u1 - u) / 1000. / ((atm->lat[ip] >= 0 ? -1 : 1) * deg2dy(1.));
00833
00834
00836
00837
          /* Potential temperature gradient... */
00838
          p1 = 0.85 * atm->p[ip];
         00839
00840
00841
00842
           / (100. * (p1 - atm->p[ip]));
00843
          /* Calculate PV... */
00844
         atm->q[ctl->qnt_pv][ip] = -1e6 * G0 * vort * grad;
00845
00846
00847
00848
        /* Calculate T_ice (Marti and Mauersberger, 1993)... */
00849
        if (ctl->qnt_tice >= 0)
00850
         atm->q[ctl->qnt\_tice][ip] = -2663.5
            / (log10(4e-6 * atm->p[ip] * 100.) - 12.537);
00851
00852
00853
        /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
        if (ctl->qnt_tnat >= 0) {
00855
         p_hno3 = 9e-9 * atm->p[ip] / 1.333224;
00856
          p_h2o = 4e-6 * atm->p[ip] / 1.333224;
         term1 = 38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o);
term2 = 0.009179 - 0.00088 * log10(p_h2o);
00857
00858
00859
         b = term1 / term2;
00860
          c = -11397.0 / term2;
00861
          x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
          x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
00862
00863
          if (x1 > 0)
00864
           atm->q[ctl->qnt_tnat][ip] = x1;
00865
          if (x2 > 0)
00866
           atm->q[ctl->qnt_tnat][ip] = x2;
00867
00868 }
00869
00871
00872 void module_position(
00873
       met_t * met0,
00874
        met_t * met1,
00875
        atm_t * atm,
00876
       int ip) {
00877
00878
       double ps:
00879
00880
        /* Calculate modulo... */
       atm->lon[ip] = fmod(atm->lon[ip], 360);
atm->lat[ip] = fmod(atm->lat[ip], 360);
00881
00882
00883
00884
        /* Check latitude... */
        while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
         if (atm->lat[ip] > 90) {
   atm->lat[ip] = 180 - atm->lat[ip];
00886
00887
00888
            atm->lon[ip] += 180;
00889
         if (atm->lat[ip] < -90) {
  atm->lat[ip] = -180 - atm->lat[ip];
  atm->lon[ip] += 180;
00890
00891
00892
00893
00894
00895
00896
        /* Check longitude... */
       while (atm->lon[ip] < -180)
atm->lon[ip] += 360;
00897
00898
00899
        while (atm->lon[ip] >= 180)
00900
         atm->lon[ip] -= 360;
00901
       00902
00903
00904
00905
                        NULL, NULL, NULL, NULL, NULL);
00906
00907
        /* Check pressure... */
       if (atm->p[ip] > ps)
00908
       atm->p[ip] = ps;
else if (atm->p[ip] < met0->p[met0->np - 1])
00909
00911
         atm \rightarrow p[ip] = met0 \rightarrow p[met0 \rightarrow np - 1];
00912 }
00913
00915
```

```
00916 void module_sedi(
       ctl_t * ctl,
met_t * met0,
00917
00918
        met_t * met1,
00919
        atm_t * atm,
00920
00921
        int ip.
00922
        double dt) {
00923
00924
        /\star Coefficients for Cunningham slip-flow correction (Kasten, 1968): \star/
00925
        const double A = 1.249, B = 0.42, C = 0.87;
00926
        /* Specific gas constant for dry air [J/(kg K)]: */
00927
       const double R = 287.058;
00928
00929
00930
        /\star Average mass of an air molecule [kg/molec]: \star/
00931
       const double m = 4.8096e-26;
00932
00933
        double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
00934
00935
        /* Check if parameters are available... */
00936
       if (ctl->qnt_r < 0 || ctl->qnt_rho < 0)
00937
          return;
00938
00939
        /* Convert units... */
00940
       p = 100 * atm -> p[ip];
        r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
00941
00942
        rho_p = atm->q[ctl->qnt_rho][ip];
00943
00944
       /* Get temperature... */
00945
       intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00946
                         atm->lat[ip], NULL, &T, NULL, NULL, NULL, NULL, NULL);
00947
00948
        /\star Density of dry air... \star/
00949
       rho = p / (R * T);
00950
       /* Dynamic viscosity of air... */
eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
00951
00953
        /* Thermal velocity of an air molecule... */ v = sqrt(8 * GSL_CONST_MKSA_BOLTZMANN * T / (M_PI * m));
00954
00955
00956
00957
        /* Mean free path of an air molecule... */
00958
        lambda = 2 * eta / (rho * v);
00959
00960
        /* Knudsen number for air... */
00961
        K = lambda / r_p;
00962
00963
        /* Cunningham slip-flow correction... */
00964
        G = 1 + K * (A + B * exp(-C / K));
00965
00966
        /* Sedimentation (fall) velocity... */
00967
        v_p =
00968
          2. * gsl_pow_2(r_p) * (rho_p -
00969
                                  rho) * GSL_CONST_MKSA_GRAV_ACCEL / (9. * eta) * G;
00970
00971
        /* Calculate pressure change... */
00972
       atm - p[ip] + = dz2dp(v_p * dt / 1000., atm - p[ip]);
00973 }
00974
00976
00977 void write_output(
00978
       const char *dirname,
00979
        ctl_t * ctl,
00980
        met_t * met0,
        met_t * met1,
00981
        atm_t * atm,
00982
00983
       double t) {
00984
00985
       char filename[LEN];
00986
00987
       double r;
00988
00989
        int year, mon, day, hour, min, sec;
00990
00991
        /* Get time... */
00992
        jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
00993
00994
        /* Write atmospheric data... */
if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
00995
         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
00996
00997
                  dirname, ctl->atm_basename, year, mon, day, hour, min);
00998
          write_atm(filename, ctl, atm, t);
00999
       }
01000
01001
       /* Write gridded data... */
```

```
if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
        sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.tab", dirname, ctl->grid_basename, year, mon, day, hour, min);
01004
          write_grid(filename, ctl, met0, met1, atm, t);
01005
01006
01007
01008
        /* Write CSI data... */
01009
        if (ctl->csi_basename[0] != '-') {
        sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01010
01011
          write_csi(filename, ctl, atm, t);
01012
01013
        /* Write profile data... */
if (ctl->prof_basename[0] != '-') {
01014
01015
        sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
write_prof(filename, ctl, met0, met1, atm, t);
01016
01017
01018
01019
01020
        /* Write station data... */
sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01022
01023
          write_station(filename, ctl, atm, t);
01024 }
01025 }
```

5.33 wind.c File Reference

Create meteorological data files with synthetic wind fields.

Functions

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

5.33.1 Detailed Description

Create meteorological data files with synthetic wind fields.

Definition in file wind.c.

5.33.2 Function Documentation

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

Definition at line 173 of file wind.c.

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

Definition at line 41 of file wind.c.

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

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

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

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

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

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

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

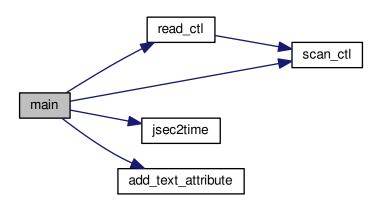
u1 = scan_ctl(argv[1], argc, argv, "WIND_UO", -1, "38.587660177302", NULL);

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

5.34 wind.c 183

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

Here is the call graph for this function:



5.34 wind.c

00001 /*

```
This file is part of MPTRAC.
00003
00004
         MPTRAC is free software: you can redistribute it and/or modify
         it under the terms of the GNU General Public License as published by
00005
         the Free Software Foundation, either version 3 of the License, or
00006
00007
         (at your option) any later version.
00008
00009
         MPTRAC is distributed in the hope that it will be useful,
00010
         but WITHOUT ANY WARRANTY; without even the implied warranty of
00011
         MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
         GNU General Public License for more details.
00013
00014
         You should have received a copy of the GNU General Public License
00015
         along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>.
00016
00017
         Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /*
00028
          Functions...
00029
00030
00031 void add_text_attribute(
        int ncid,
00033
         char *varname,
00034
         char *attrname,
00035
         char *text);
00036
00037 /*
00038
         Main...
00039
00040
00041 int main(
00042
         int argc,
00043
         char *argv[]) {
00044
00045
         ctl t ctl;
00046
00047
         static char filename[LEN];
00048
00049
         static double r, t0, z0, z1, dataLon[EX], dataLat[EY], dataZ[EP],
00050
           u0, u1, alpha;
00051
00052
         static float dataT[EP * EY * EX], dataU[EP * EY * EX], dataV[EP * EY * EX],
00053
           dataW[EP * EY * EX];
00054
00055
         static int ncid, dims[4], timid, levid, latid, lonid, tid, uid, wid, wid,
00056
           idx, ix, iy, iz, nx, ny, nz, year, mon, day, hour, min, sec;
00057
00058
         /* Check arguments... */
00059
         if (argc < 3)</pre>
00060
           ERRMSG("Give parameters: <ctl> <metbase>");
00061
00062
         /* Read control parameters... */
         read_ctl(argv[1], argc, argv, &ctl);
00063
         read_ctl(argv[1], argc, argv, &ctl);
t0 = scan_ctl(argv[1], argc, argv, "WIND_T0", -1, "0", NULL);
nx = (int) scan_ctl(argv[1], argc, argv, "WIND_NX", -1, "360", NULL);
ny = (int) scan_ctl(argv[1], argc, argv, "WIND_NY", -1, "181", NULL);
nz = (int) scan_ctl(argv[1], argc, argv, "WIND_NZ", -1, "61", NULL);
z0 = scan_ctl(argv[1], argc, argv, "WIND_Z0", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "WIND_Z1", -1, "60", NULL);
00064
00065
00066
00067
00068
00069
         u0 = scan_ctl(argy[1], argc, argv, "WIND_U0", -1, "38.587660177302", NULL);
u1 = scan_ctl(argv[1], argc, argv, "WIND_U1", -1, "38.587660177302", NULL);
00070
00071
         alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);
00072
00073
00074
         /* Check dimensions... */
00075
         if (nx < 1 || nx > EX)
00076
           ERRMSG("Set 1 <= NX <= MAX!");</pre>
           f (ny < 1 || ny > EY)
ERRMSG("Set 1 <= NY <= MAX!");
00077
00078
         if (nz < 1 || nz > EP)
   ERRMSG("Set 1 <= NZ <= MAX!");</pre>
00079
00080
00081
         /* Get time... */
00082
00083
          jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
00084
         t0 = year * 10000. + mon * 100. + day + hour / 24.;
00085
         /* Set filename... */
sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", argv[2], year, mon, day, hour);
00086
00087
00088
          /* Create netCDF file...
00089
00090
         NC(nc_create(filename, NC_CLOBBER, &ncid));
00091
         /* Create dimensions... */
NC(nc_def_dim(ncid, "time", 1, &dims[0]));
00092
00093
```

5.34 wind.c 185

```
NC(nc_def_dim(ncid, "lev", (size_t) nz, &dims[1]));
NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[2]));
NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[3]));
00095
00096
00097
           /* Create variables... */
NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));
NC(nc_def_var(ncid, "lev", NC_DOUBLE, 1, &dims[1], &levid));
NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[2], &latid));
00098
00099
00100
00101
           NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[3], &lonid));
00102
           NC(nc_def_var(ncid, "T", NC_FLOAT, 4, &dims[0], &tid));
NC(nc_def_var(ncid, "U", NC_FLOAT, 4, &dims[0], &uid));
NC(nc_def_var(ncid, "V", NC_FLOAT, 4, &dims[0], &vid));
00103
00104
00105
00106
           NC(nc_def_var(ncid, "W", NC_FLOAT, 4, &dims[0], &wid));
00107
00108
            /* Set attributes... */
           /* Set attributes... */
add_text_attribute(ncid, "time", "long_name", "time");
add_text_attribute(ncid, "time", "units", "day as %Y%m%d.%f");
add_text_attribute(ncid, "lon", "long_name", "longitude");
add_text_attribute(ncid, "lon", "units", "degrees_east");
add_text_attribute(ncid, "lat", "long_name", "latitude");
add_text_attribute(ncid, "lat", "units", "degrees_north");
add_text_attribute(ncid, "lev", "long_name", "air_pressure");
add_text_attribute(ncid, "lev", "units", "Pa");
add_text_attribute(ncid, "T", "units", "Fa");
add_text_attribute(ncid, "T", "units", "K");
add_text_attribute(ncid, "T", "units", "K");
add_text_attribute(ncid, "T", "units", "R");
add_text_attribute(ncid, "T", "units", "R");
add_text_attribute(ncid, "T", "units", "R");
add_text_attribute(ncid, "T", "units", "R");
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
           add_text_attribute(ncid, "U", "long_name", "U velocity");
add_text_attribute(ncid, "U", "units", "m s**-1");
00119
00120
            add_text_attribute(ncid, "V", "long_name", "V velocity");
00121
           add_text_attribute(ncid, "V", "units", "m s**-1");
add_text_attribute(ncid, "W", "long_name", "Vertical velocity");
add_text_attribute(ncid, "W", "long_name", "Vertical velocity");
00122
00123
00124
00125
00126
            /* End definition... */
00127
           NC(nc_enddef(ncid));
00128
            /* Set coordinates... */
00129
           for (ix = 0; ix < nx; ix++)
00130
             dataLon[ix] = 360.0 / nx * (double) ix;
00131
00132
           for (iy = 0; iy < ny; iy++)
00133
              dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
00134
            for (iz = 0; iz < nz; iz++)
              dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));
00135
00136
00137
            /* Write coordinates... */
00138
           NC(nc_put_var_double(ncid, timid, &t0));
00139
           NC(nc_put_var_double(ncid, levid, dataZ));
00140
           NC(nc_put_var_double(ncid, lonid, dataLon));
00141
           NC(nc_put_var_double(ncid, latid, dataLat));
00142
00143
            /* Create wind fields (Williamson et al., 1992)... */
00144
           for (ix = 0; ix < nx; ix++)
00145
              for (iy = 0; iy < ny; iy++)</pre>
00146
                  for (iz = 0; iz < nz; iz++) {</pre>
                     idx = (iz * ny + iy) * nx + ix;
00147
                     dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
00148
                                                      * (cos(dataLat[iy] * M_PI / 180.0)

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

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

Index

add_text_attribute	ctl_t, 14
wind.c, 181	ctl_t, 5
atm_basename	atm_basename, 12
ctl_t, 12	atm_dt_out, 13
atm_dt_out	atm_filter, 13
ctl_t, 13	atm_gpfile, 12
atm_filter	balloon, 11
ctl_t, 13	csi_basename, 13
atm_gpfile	csi_dt_out, 13
ctl_t, 12	csi_lat0, 14
atm_t, 3	csi_lat1, 14
lat, 4	csi_lon0, 14
lon, 4	csi_lon1, 14
np, 4	csi_modmin, 13
p, 4	csi_nx, 14
q, 4	csi_ny, 14
time, 4	csi_nz, 13
up, 4	csi_obsfile, 13
vp, 4	csi_obsmin, 13
wp, 5	csi_z0, 13
balloon	csi_z1, 14
ctl t, 11	direction, 11
_	dt_met, 11
cart2geo	dt_mod, 11
libtrac.c, 40	grid_basename, 14
libtrac.h, 90	grid_dt_out, 15 grid_gpfile, 14
center.c, 21	grid_gprile, 14 grid_lat0, 16
main, 21	grid_lat1, 16
csi_basename	grid_lon0, 15
ctl_t, 13	grid_lon1, 15
csi_dt_out	grid_nx, 15
ctl_t, 13	grid_ny, 15
csi_lat0 ctl t, 14	grid_nz, 15
csi lat1	grid_sparse, 15
ctl t, 14	grid_z0, 15
csi_lon0	grid_z1, 15
ctl_t, 14	isosurf, 11
csi lon1	met_np, 11
ctl_t, 14	met_p, 11
csi_modmin	nq, <mark>8</mark>
ctl_t, 13	prof_basename, 16
csi_nx	prof_lat0, 17
ctl_t, 14	prof_lat1, 17
csi_ny	prof_lon0, 16
ctl_t, 14	prof_lon1, 17
csi_nz	prof_nx, 16
ctl_t, 13	prof_ny, 17
csi_obsfile	prof_nz, 16
ctl_t, 13	prof_obsfile, 16
csi_obsmin	prof_z0, 16
ctl_t, 13	prof_z1, 16 qnt_format, 9
csi_z0	qnt_lormat, 9 qnt_h2o, 10
ctl_t, 13 csi_z1	qnt_m, 9
V31_2 I	qπ_m, σ

qnt_name, 8	geo2cart
qnt_o3, 10	libtrac.c, 41
qnt_p, 9	libtrac.h, 92
qnt_ps, 9	get_met
qnt_pv, 10	libtrac.c, 42
qnt_r, 9	libtrac.h, 92
qnt_rho, 9	get_met_help
qnt_stat, 10	libtrac.c, 43
qnt_t, 9	libtrac.h, 93
qnt_theta, 10	grid_basename
ant tice, 10	ctl t, 14
qnt_tnat, 10	grid_dt_out
qnt_u, 9	ctl_t, 15
• —	
qnt_unit, 9	grid_gpfile
qnt_v, 10	ctl_t, 14
qnt_w, 10	grid_lat0
stat_basename, 17	ctl_t, 16
stat_lat, 17	grid_lat1
stat_lon, 17	ctl_t, 16
stat_r, 17	grid_lon0
t_start, 11	ctl_t, 15
t_stop, 11	grid_lon1
tdec_strat, 12	ctl_t, 15
tdec_trop, 12	grid_nx
turb_dx_strat, 12	ctl_t, 15
turb_dx_trop, 12	grid_ny
turb_dz_strat, 12	ctl_t, 15
turb_dz_trop, 12	grid_nz
turb_meso, 12	ctl_t, 15
	grid_sparse
deg2dx	ctl_t, 15
libtrac.c, 40	
libtrac.h, 90	grid_z0
deg2dy	ctl_t, 15
libtrac.c, 40	grid_z1
libtrac.h, 91	ctl_t, 15
direction	h0-
	h2o
ctl_t, 11	met_t, 20
dist.c, 25	: :: 00
main, 25	init.c, 33
dp2dz	main, 34
libtrac.c, 41	init_simtime
libtrac.h, 91	trac.c, 155
dt_met	intpol_met_2d
ctl_t, 11	libtrac.c, 43
dt_mod	libtrac.h, <mark>93</mark>
ctl_t, 11	intpol_met_3d
dx2deg	libtrac.c, 44
libtrac.c, 41	libtrac.h, 94
libtrac.h, 91	intpol_met_space
dy2deg	libtrac.c, 44
libtrac.c, 41	libtrac.h, 94
libtrac.h, 91	intpol_met_time
dz2dp	libtrac.c, 45
libtrac.c, 41	libtrac.h, 95
	isosurf
libtrac.h, 91	
extract.c, 31	ctl_t, 11
	jsec2time
main, 31	Jacozume

libtrac.c, 46	read_ctl, 97
libtrac.h, 96	read_met, 100
jsec2time.c, 37	read_met_extrapolate, 102
main, 37	read_met_help, 103
	read_met_ml2pl, 103
lat	read_met_periodic, 104
atm_t, 4	scan_ctl, 104
met_t, 19	time2jsec, 105
libtrac.c, 38	timer, 106
cart2geo, 40	tropopause, 106
deg2dx, 40	write atm, 108
deg2dy, 40	write_csi, 109
dp2dz, 41	write_grid, 111
dx2deg, 41	write_prof, 113
dy2deg, 41	write_station, 116
dz2dp, 41	locate
geo2cart, 41	libtrac.c, 46
get_met, 42	libtrac.h, 96
get_met_help, 43	lon
intpol_met_2d, 43	atm t, 4
intpol_met_3d, 44	met_t, 19
intpol_met_space, 44	
intpol met time, 45	main
jsec2time, 46	center.c, 21
locate, 46	dist.c, 25
read_atm, 47	extract.c, 31
read_ctl, 47	init.c, 34
read_met, 50	jsec2time.c, 37
read_met_extrapolate, 52	match.c, 124
read_met_help, 53	met_map.c, 129
read_met_ml2pl, 53	met_prof.c, 132
read_met_periodic, 54	met_sample.c, 137
scan ctl, 54	met_zm.c, 141
time2jsec, 55	smago.c, 145
timer, 56	split.c, 149
tropopause, 56	time2jsec.c, 153
write atm, 58	trac.c, 165
write_csi, 59	wind.c, 181
write grid, 61	match.c, 124
write_prof, 63	main, 124
write_station, 66	met_map.c, 128
libtrac.h, 89	main, 129
cart2geo, 90	met np
deg2dx, 90	ctl_t, 11
deg2dy, 91	met p
dp2dz, 91	–
dx2deg, 91	ctl_t, 11
dy2deg, 91 dy2deg, 91	met_prof.c, 132
· · · · ·	main, 132
dz2dp, 91	met_sample.c, 137
geo2cart, 92	main, 137
get_met, 92	met_t, 18
get_met_help, 93	h2o, 20
intpol_met_2d, 93	lat, 19
intpol_met_3d, 94	lon, 19
intpol_met_space, 94	np, 19
intpol_met_time, 95	nx, 19
jsec2time, 96	ny, 19
locate, 96	03, 20
read_atm, 97	p, 19

	11 1 40
pl, 20	ctl_t, 16
ps, 19	prof_obsfile
t, 20	ctl_t, 16
time, 19	prof_z0
u, 20	ctl_t, 16
v, 20	prof z1
w, 20	ctl_t, 16
met_zm.c, 140	ps
main, 141	met_t, 19
module_advection	
trac.c, 155	q
module_decay	atm_t, 4
trac.c, 155	qnt_format
module_diffusion_meso	ctl_t, 9
	qnt_h2o
trac.c, 156	•
module_diffusion_turb	ctl_t, 10
trac.c, 158	qnt_m
module_isosurf	ctl_t, 9
trac.c, 159	qnt_name
module_meteo	ctl t, 8
trac.c, 161	gnt o3
	ctl_t, 10
module_position	
trac.c, 162	qnt_p
module_sedi	ctl_t, 9
trac.c, 163	qnt_ps
	ctl_t, 9
np	qnt_pv
atm_t, 4	ctl_t, 10
met_t, 19	qnt_r
	ctl_t, 9
nq	
ctl_t, 8	qnt_rho
nx	ctl_t, 9
met_t, 19	qnt_stat
ny	ctl_t, 10
met t, 19	qnt_t
<u> </u>	ctl_t, 9
03	ont theta
03 met t 20	qnt_theta
o3 met_t, 20	ctl_t, 10
met_t, 20	ctl_t, 10 qnt_tice
met_t, 20 p	ctl_t, 10 qnt_tice ctl_t, 10
met_t, 20 p atm_t, 4	ctl_t, 10 qnt_tice
met_t, 20 p	ctl_t, 10 qnt_tice ctl_t, 10
met_t, 20 p atm_t, 4	ctl_t, 10 qnt_tice ctl_t, 10 qnt_tnat ctl_t, 10
met_t, 20 p atm_t, 4 met_t, 19	ctl_t, 10 qnt_tice ctl_t, 10 qnt_tnat ctl_t, 10 qnt_u
met_t, 20 p atm_t, 4 met_t, 19 pl met_t, 20	ctl_t, 10 qnt_tice ctl_t, 10 qnt_tnat ctl_t, 10 qnt_u ctl_t, 9
met_t, 20 p atm_t, 4 met_t, 19 pl met_t, 20 prof_basename	ctl_t, 10 qnt_tice ctl_t, 10 qnt_tnat ctl_t, 10 qnt_u ctl_t, 9 qnt_unit
met_t, 20 p atm_t, 4 met_t, 19 pl met_t, 20 prof_basename ctl_t, 16	ctl_t, 10 qnt_tice ctl_t, 10 qnt_tnat ctl_t, 10 qnt_u ctl_t, 9 qnt_unit ctl_t, 9
met_t, 20 p atm_t, 4 met_t, 19 pl met_t, 20 prof_basename ctl_t, 16 prof_lat0	ctl_t, 10 qnt_tice ctl_t, 10 qnt_tnat ctl_t, 10 qnt_u ctl_t, 9 qnt_unit ctl_t, 9 qnt_unit
met_t, 20 p atm_t, 4 met_t, 19 pl met_t, 20 prof_basename ctl_t, 16 prof_lat0 ctl_t, 17	ctl_t, 10 qnt_tice ctl_t, 10 qnt_tnat ctl_t, 10 qnt_u ctl_t, 9 qnt_unit ctl_t, 9
met_t, 20 p atm_t, 4 met_t, 19 pl met_t, 20 prof_basename ctl_t, 16 prof_lat0 ctl_t, 17 prof_lat1	ctl_t, 10 qnt_tice ctl_t, 10 qnt_tnat ctl_t, 10 qnt_u ctl_t, 9 qnt_unit ctl_t, 9 qnt_unit
met_t, 20 p atm_t, 4 met_t, 19 pl met_t, 20 prof_basename ctl_t, 16 prof_lat0 ctl_t, 17	ctl_t, 10 qnt_tice ctl_t, 10 qnt_tnat ctl_t, 10 qnt_u ctl_t, 9 qnt_unit ctl_t, 9 qnt_v ctl_t, 10
met_t, 20 p atm_t, 4 met_t, 19 pl met_t, 20 prof_basename ctl_t, 16 prof_lat0 ctl_t, 17 prof_lat1	ctl_t, 10 qnt_tice ctl_t, 10 qnt_tnat ctl_t, 10 qnt_u ctl_t, 9 qnt_unit ctl_t, 9 qnt_v ctl_t, 10 qnt_v ctl_t, 10 qnt_w
met_t, 20 p atm_t, 4 met_t, 19 pl met_t, 20 prof_basename ctl_t, 16 prof_lat0 ctl_t, 17 prof_lat1 ctl_t, 17 prof_lon0	ctl_t, 10 qnt_tice ctl_t, 10 qnt_tnat ctl_t, 10 qnt_u ctl_t, 9 qnt_unit ctl_t, 9 qnt_v ctl_t, 10 qnt_w ctl_t, 10
met_t, 20 p atm_t, 4 met_t, 19 pl met_t, 20 prof_basename ctl_t, 16 prof_lat0 ctl_t, 17 prof_lat1 ctl_t, 17 prof_lon0 ctl_t, 16	ctl_t, 10 qnt_tice ctl_t, 10 qnt_tnat ctl_t, 10 qnt_u ctl_t, 9 qnt_unit ctl_t, 9 qnt_v ctl_t, 10 qnt_w ctl_t, 10 qnt_w ctl_t, 10 read_atm
met_t, 20 p atm_t, 4 met_t, 19 pl met_t, 20 prof_basename ctl_t, 16 prof_lat0 ctl_t, 17 prof_lat1 ctl_t, 17 prof_lon0 ctl_t, 16 prof_lon1	ctl_t, 10 qnt_tice ctl_t, 10 qnt_tnat ctl_t, 10 qnt_u ctl_t, 9 qnt_unit ctl_t, 9 qnt_v ctl_t, 10 qnt_w ctl_t, 10 qnt_w ctl_t, 10 read_atm libtrac.c, 47
met_t, 20 p atm_t, 4 met_t, 19 pl met_t, 20 prof_basename ctl_t, 16 prof_lat0 ctl_t, 17 prof_lat1 ctl_t, 17 prof_lon0 ctl_t, 16 prof_lon1 ctl_t, 17	ctl_t, 10 qnt_tice ctl_t, 10 qnt_tnat ctl_t, 10 qnt_u ctl_t, 9 qnt_unit ctl_t, 9 qnt_v ctl_t, 10 qnt_w ctl_t, 10 read_atm libtrac.c, 47 libtrac.h, 97
met_t, 20 p atm_t, 4 met_t, 19 pl met_t, 20 prof_basename ctl_t, 16 prof_lat0 ctl_t, 17 prof_lat1 ctl_t, 17 prof_lon0 ctl_t, 16 prof_lon1 ctl_t, 17 prof_lon1	ctl_t, 10 qnt_tice ctl_t, 10 qnt_tat ctl_t, 10 qnt_u ctl_t, 9 qnt_unit ctl_t, 9 qnt_v ctl_t, 10 qnt_w ctl_t, 10 read_atm libtrac.c, 47 libtrac.h, 97 read_ctl
met_t, 20 p atm_t, 4 met_t, 19 pl met_t, 20 prof_basename ctl_t, 16 prof_lat0 ctl_t, 17 prof_lat1 ctl_t, 17 prof_lon0 ctl_t, 16 prof_lon1 ctl_t, 17	ctl_t, 10 qnt_tice ctl_t, 10 qnt_tnat ctl_t, 10 qnt_u ctl_t, 9 qnt_unit ctl_t, 9 qnt_v ctl_t, 10 qnt_w ctl_t, 10 read_atm libtrac.c, 47 libtrac.c, 47
met_t, 20 p atm_t, 4 met_t, 19 pl met_t, 20 prof_basename ctl_t, 16 prof_lat0 ctl_t, 17 prof_lat1 ctl_t, 17 prof_lon0 ctl_t, 16 prof_lon1 ctl_t, 17 prof_lon1	ctl_t, 10 qnt_tice ctl_t, 10 qnt_tat ctl_t, 10 qnt_u ctl_t, 9 qnt_unit ctl_t, 9 qnt_v ctl_t, 10 qnt_w ctl_t, 10 read_atm libtrac.c, 47 libtrac.h, 97 read_ctl
met_t, 20 p atm_t, 4 met_t, 19 pl met_t, 20 prof_basename ctl_t, 16 prof_lat0 ctl_t, 17 prof_lat1 ctl_t, 17 prof_lon0 ctl_t, 16 prof_lon1 ctl_t, 17 prof_lon1 ctl_t, 17	ctl_t, 10 qnt_tice ctl_t, 10 qnt_tnat ctl_t, 10 qnt_u ctl_t, 9 qnt_unit ctl_t, 9 qnt_v ctl_t, 10 qnt_w ctl_t, 10 read_atm libtrac.c, 47 libtrac.c, 47
met_t, 20 p atm_t, 4 met_t, 19 pl met_t, 20 prof_basename ctl_t, 16 prof_lat0 ctl_t, 17 prof_lat1 ctl_t, 17 prof_lon0 ctl_t, 16 prof_lon1 ctl_t, 17 prof_nx ctl_t, 16 prof_ny	ctl_t, 10 qnt_tice ctl_t, 10 qnt_tnat ctl_t, 10 qnt_u ctl_t, 9 qnt_unit ctl_t, 9 qnt_v ctl_t, 10 qnt_w ctl_t, 10 read_atm libtrac.c, 47 libtrac.h, 97 read_ctl libtrac.h, 97

libtrac.h, 100	module_meteo, 161
read_met_extrapolate	module_position, 162
libtrac.c, 52	module_sedi, 163
libtrac.h, 102	write_output, 164
read_met_help	tropopause
libtrac.c, 53	libtrac.c, 56
libtrac.h, 103	libtrac.h, 106
read_met_ml2pl	turb_dx_strat
libtrac.c, 53	ctl_t, 12
libtrac.h, 103	turb_dx_trop
read_met_periodic	ctl_t, 12
libtrac.c, 54	turb_dz_strat
libtrac.h, 104	ctl_t, 12
	turb_dz_trop
scan_ctl	ctl_t, 12
libtrac.c, 54	turb_meso
libtrac.h, 104	ctl_t, 12
smago.c, 145	
main, 145	u
split.c, 148	met_t, 20
main, 149	up
stat_basename	atm t, 4
 ctl_t, 17	<u> </u>
stat lat	V
ctl_t, 17	met t, 20
	vp _ ^
stat_lon	atm t, 4
ctl_t, 17	<u> </u>
stat_r	w
ctl_t, 17	met_t, 20
i	
t	wind.c, 181
met_t, 20	wind.c, 181 add_text_attribute, 181
met_t, 20 t_start	wind.c, 181 add_text_attribute, 181 main, 181
met_t, 20	wind.c, 181 add_text_attribute, 181 main, 181 wp
met_t, 20 t_start	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5
met_t, 20 t_start ctl_t, 11	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm
met_t, 20 t_start ctl_t, 11 t_stop	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop ctl_t, 12 time	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi libtrac.c, 59
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop ctl_t, 12 time atm_t, 4	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi libtrac.c, 59 libtrac.h, 109
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop ctl_t, 12 time atm_t, 4 met_t, 19	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi libtrac.c, 59 libtrac.h, 109 write_grid
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop ctl_t, 12 time atm_t, 4 met_t, 19 time2jsec	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi libtrac.c, 59 libtrac.h, 109 write_grid libtrac.c, 61 libtrac.h, 111
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop ctl_t, 12 time atm_t, 4 met_t, 19 time2jsec libtrac.c, 55	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi libtrac.c, 59 libtrac.h, 109 write_grid libtrac.c, 61 libtrac.h, 111 write_output
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop ctl_t, 12 time atm_t, 4 met_t, 19 time2jsec libtrac.c, 55 libtrac.h, 105	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi libtrac.c, 59 libtrac.h, 109 write_grid libtrac.c, 61 libtrac.h, 111 write_output trac.c, 164
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop ctl_t, 12 time atm_t, 4 met_t, 19 time2jsec libtrac.c, 55 libtrac.h, 105 time2jsec.c, 152	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi libtrac.c, 59 libtrac.h, 109 write_grid libtrac.c, 61 libtrac.h, 111 write_output trac.c, 164 write_prof
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop ctl_t, 12 time atm_t, 4 met_t, 19 time2jsec libtrac.c, 55 libtrac.h, 105 time2jsec.c, 152 main, 153	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi libtrac.c, 59 libtrac.h, 109 write_grid libtrac.c, 61 libtrac.h, 111 write_output trac.c, 164 write_prof libtrac.c, 63
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop ctl_t, 12 time atm_t, 4 met_t, 19 time2jsec libtrac.c, 55 libtrac.h, 105 time2jsec.c, 152 main, 153 timer	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi libtrac.c, 59 libtrac.h, 109 write_grid libtrac.c, 61 libtrac.h, 111 write_output trac.c, 164 write_prof libtrac.c, 63 libtrac.h, 113
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop ctl_t, 12 time atm_t, 4 met_t, 19 time2jsec libtrac.c, 55 libtrac.h, 105 time2jsec.c, 152 main, 153 timer libtrac.c, 56	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi libtrac.c, 59 libtrac.h, 109 write_grid libtrac.c, 61 libtrac.h, 111 write_output trac.c, 164 write_prof libtrac.h, 113 write_station
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop ctl_t, 12 time atm_t, 4 met_t, 19 time2jsec libtrac.c, 55 libtrac.h, 105 time2jsec.c, 152 main, 153 timer	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi libtrac.c, 59 libtrac.h, 109 write_grid libtrac.c, 61 libtrac.h, 111 write_output trac.c, 164 write_prof libtrac.h, 113 write_station libtrac.c, 66
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop ctl_t, 12 time atm_t, 4 met_t, 19 time2jsec libtrac.c, 55 libtrac.h, 105 time2jsec.c, 152 main, 153 timer libtrac.c, 56	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi libtrac.c, 59 libtrac.h, 109 write_grid libtrac.c, 61 libtrac.h, 111 write_output trac.c, 164 write_prof libtrac.h, 113 write_station
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop ctl_t, 12 time atm_t, 4 met_t, 19 time2jsec libtrac.c, 55 libtrac.h, 105 time2jsec.c, 152 main, 153 timer libtrac.c, 56 libtrac.h, 106	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi libtrac.c, 59 libtrac.h, 109 write_grid libtrac.c, 61 libtrac.h, 111 write_output trac.c, 164 write_prof libtrac.h, 113 write_station libtrac.c, 66
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop ctl_t, 12 time atm_t, 4 met_t, 19 time2jsec libtrac.c, 55 libtrac.h, 105 time2jsec.c, 152 main, 153 timer libtrac.c, 56 libtrac.h, 106 trac.c, 154	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi libtrac.c, 59 libtrac.h, 109 write_grid libtrac.c, 61 libtrac.h, 111 write_output trac.c, 164 write_prof libtrac.h, 113 write_station libtrac.c, 66
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop ctl_t, 12 time atm_t, 4 met_t, 19 time2jsec libtrac.c, 55 libtrac.h, 105 time2jsec.c, 152 main, 153 timer libtrac.c, 56 libtrac.h, 106 trac.c, 154 init_simtime, 155	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi libtrac.c, 59 libtrac.h, 109 write_grid libtrac.c, 61 libtrac.h, 111 write_output trac.c, 164 write_prof libtrac.h, 113 write_station libtrac.c, 66
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop ctl_t, 12 time atm_t, 4 met_t, 19 time2jsec libtrac.c, 55 libtrac.h, 105 time2jsec.c, 152 main, 153 timer libtrac.c, 56 libtrac.h, 106 trac.c, 154 init_simtime, 155 main, 165 module_advection, 155	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi libtrac.c, 59 libtrac.h, 109 write_grid libtrac.c, 61 libtrac.h, 111 write_output trac.c, 164 write_prof libtrac.h, 113 write_station libtrac.c, 66
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop ctl_t, 12 time atm_t, 4 met_t, 19 time2jsec libtrac.c, 55 libtrac.h, 105 time2jsec.c, 152 main, 153 timer libtrac.c, 56 libtrac.h, 106 trac.c, 154 init_simtime, 155 main, 165 module_advection, 155 module_decay, 155	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi libtrac.c, 59 libtrac.h, 109 write_grid libtrac.c, 61 libtrac.h, 111 write_output trac.c, 164 write_prof libtrac.h, 113 write_station libtrac.c, 66
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop ctl_t, 12 time atm_t, 4 met_t, 19 time2jsec libtrac.c, 55 libtrac.h, 105 time2jsec.c, 152 main, 153 timer libtrac.c, 56 libtrac.h, 106 trac.c, 154 init_simtime, 155 madule_advection, 155 module_decay, 155 module_diffusion_meso, 156	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi libtrac.c, 59 libtrac.h, 109 write_grid libtrac.c, 61 libtrac.h, 111 write_output trac.c, 164 write_prof libtrac.h, 113 write_station libtrac.c, 66
met_t, 20 t_start ctl_t, 11 t_stop ctl_t, 11 tdec_strat ctl_t, 12 tdec_trop ctl_t, 12 time atm_t, 4 met_t, 19 time2jsec libtrac.c, 55 libtrac.h, 105 time2jsec.c, 152 main, 153 timer libtrac.c, 56 libtrac.h, 106 trac.c, 154 init_simtime, 155 main, 165 module_advection, 155 module_decay, 155	wind.c, 181 add_text_attribute, 181 main, 181 wp atm_t, 5 write_atm libtrac.c, 58 libtrac.h, 108 write_csi libtrac.c, 59 libtrac.h, 109 write_grid libtrac.c, 61 libtrac.h, 111 write_output trac.c, 164 write_prof libtrac.h, 113 write_station libtrac.c, 66