# **MPTRAC**

Generated by Doxygen 1.8.11

ii CONTENTS

# Contents

1	Main	n Page		2	
2	2 Data Structure Index				
	2.1	Data St	tructures	2	
3	File Index				
	3.1	File Lis	t	2	
4 Data Structure Documentation				3	
	4.1	atm_t S	Struct Reference	3	
		4.1.1	Detailed Description	4	
		4.1.2	Field Documentation	4	
	4.2	ctl_t St	ruct Reference	5	
		4.2.1	Detailed Description	9	
		4.2.2	Field Documentation	9	
	4.3	met_t S	Struct Reference	20	
		4.3.1	Detailed Description	21	
		4.3.2	Field Documentation	21	
5	File	Docume	entation	23	
	5.1	center.	c File Reference	23	
		5.1.1	Detailed Description	23	
		5.1.2	Function Documentation	23	
	5.2	center.	C	25	
	5.3	cluster.	c File Reference	27	
		5.3.1	Detailed Description	27	
		5.3.2	Function Documentation	27	
	5.4	cluster.	C	30	
	5.5	dist.c F	ile Reference	33	
		5.5.1	Detailed Description	33	
		5.5.2	Function Documentation	33	

5.6	dist.c	36
5.7	extract.c File Reference	38
	5.7.1 Detailed Description	39
	5.7.2 Function Documentation	39
5.8	extract.c	40
5.9	init.c File Reference	41
	5.9.1 Detailed Description	41
	5.9.2 Function Documentation	41
5.10	init.c	43
5.11	jsec2time.c File Reference	45
	5.11.1 Detailed Description	45
	5.11.2 Function Documentation	45
5.12	jsec2time.c	46
5.13	libtrac.c File Reference	46
	5.13.1 Detailed Description	48
	5.13.2 Function Documentation	48
5.14	libtrac.c	78
5.15	libtrac.h File Reference	102
	5.15.1 Detailed Description	104
	5.15.2 Function Documentation	104
5.16	libtrac.h	134
5.17	match.c File Reference	142
	5.17.1 Detailed Description	142
	5.17.2 Function Documentation	142
5.18	match.c	144
5.19	met_map.c File Reference	146
	5.19.1 Detailed Description	147
	5.19.2 Function Documentation	147
5.20	met_map.c	148
5.21	met_prof.c File Reference	150

	5.21.1 Detailed Description	150
	5.21.2 Function Documentation	150
5.22	met_prof.c	153
5.23	met_sample.c File Reference	154
	5.23.1 Detailed Description	155
	5.23.2 Function Documentation	155
5.24	met_sample.c	156
5.25	met_zm.c File Reference	157
	5.25.1 Detailed Description	158
	5.25.2 Function Documentation	158
5.26	met_zm.c	160
5.27	smago.c File Reference	162
	5.27.1 Detailed Description	162
	5.27.2 Function Documentation	162
5.28	smago.c	164
5.29	split.c File Reference	165
	5.29.1 Detailed Description	165
	5.29.2 Function Documentation	166
5.30	split.c	168
5.31	time2jsec.c File Reference	169
	5.31.1 Detailed Description	169
	5.31.2 Function Documentation	170
5.32	time2jsec.c	170
5.33	trac.c File Reference	171
	5.33.1 Detailed Description	171
	5.33.2 Function Documentation	172
5.34	trac.c	190
5.35	wind.c File Reference	206
	5.35.1 Detailed Description	206
	5.35.2 Function Documentation	206
5.36	wind.c	208

Index 213

## 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	20

## 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	23
cluster.c Clustering of trajectories	27
dist.c Calculate transport deviations of trajectories	33
extract.c Extract single trajectory from atmospheric data files	38
init.c Create atmospheric data file with initial air parcel positions	41
jsec2time.c Convert Julian seconds to date	45
libtrac.c MPTRAC library definitions	46

libtrac.h MPTRAC library declarations	102
match.c Calculate deviations between two trajectories	142
met_map.c Extract global map from meteorological data	146
met_prof.c Extract vertical profile from meteorological data	150
met_sample.c Sample meteorological data at given geolocations	154
met_zm.c Extract zonal mean from meteorological data	157
smago.c Estimate horizontal diffusivity based on Smagorinsky theory	162
split.c Split air parcels into a larger number of parcels	165
time2jsec.c Convert date to Julian seconds	169
trac.c Lagrangian particle dispersion model	171
wind.c Create meteorological data files with synthetic wind fields	206

## 4 Data Structure Documentation

## 4.1 atm\_t Struct Reference

Atmospheric data.

#include <libtrac.h>

## **Data Fields**

• int np

Number of air pacels.

• double time [NP]

Time [s].

• double p [NP]

Pressure [hPa].

• double lon [NP]

Longitude [deg].

• double lat [NP]

Latitude [deg].

double q [NQ][NP]

```
Quantitiy data (for various, user-defined attributes).
     • float up [NP]
           Zonal wind perturbation [m/s].
     • float vp [NP]
           Meridional wind perturbation [m/s].

    float wp [NP]

           Vertical velocity perturbation [hPa/s].
4.1.1 Detailed Description
Atmospheric data.
Definition at line 520 of file libtrac.h.
4.1.2 Field Documentation
4.1.2.1 int atm_t::np
Number of air pacels.
Definition at line 523 of file libtrac.h.
4.1.2.2 double atm_t::time[NP]
Time [s].
Definition at line 526 of file libtrac.h.
4.1.2.3 double atm_t::p[NP]
Pressure [hPa].
Definition at line 529 of file libtrac.h.
4.1.2.4 double atm_t::lon[NP]
Longitude [deg].
Definition at line 532 of file libtrac.h.
4.1.2.5 double atm_t::lat[NP]
Latitude [deg].
Definition at line 535 of file libtrac.h.
4.1.2.6 double atm_t::q[NQ][NP]
Quantitiy data (for various, user-defined attributes).
Definition at line 538 of file libtrac.h.
```

4.1.2.7 float atm\_t::up[NP]

Zonal wind perturbation [m/s].

```
Definition at line 541 of file libtrac.h.
4.1.2.8 float atm_t::vp[NP]
Meridional wind perturbation [m/s].
Definition at line 544 of file libtrac.h.
4.1.2.9 float atm_t::wp[NP]
Vertical velocity perturbation [hPa/s].
Definition at line 547 of file libtrac.h.
The documentation for this struct was generated from the following file:
    · libtrac.h
4.2 ctl_t Struct Reference
Control parameters.
#include <libtrac.h>
Data Fields
    • int nq
          Number of quantities.

    char qnt_name [NQ][LEN]

          Quantity names.
    char qnt_unit [NQ][LEN]
          Quantity units.

    char qnt_format [NQ][LEN]

          Quantity output format.
    • int qnt_ens
          Quantity array index for ensemble IDs.
    • int qnt_m
          Quantity array index for mass.
    int qnt_rho
          Quantity array index for particle density.
    • int qnt_r
          Quantity array index for particle radius.
    int qnt_ps
          Quantity array index for surface pressure.
    int qnt_p
          Quantity array index for pressure.
Generated by Doxygen
```

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\_tsts

Quantity array index for T\_STS.

int qnt\_tnat

Quantity array index for T\_NAT.

int qnt\_stat

Quantity array index for station flag.

• int qnt\_gw\_u750

Quantity array index for low-level zonal wind.

int qnt\_gw\_v750

Quantity array index for low-level meridional wind.

• int qnt\_gw\_sso

Quantity array index for subgrid-scale orography.

• int qnt\_gw\_var

Quantity array index for gravity wave variances.

· 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].

char met\_stage [LEN]

Command to stage meteo data.

· int isosurf

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

• char balloon [LEN]

Balloon position filename.

double turb\_dx\_trop

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

· double turb dx strat

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

double turb dz trop

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

· double turb dz strat

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

· double turb\_meso

Scaling factor for mesoscale wind fluctuations.

· double tdec\_trop

Life time of particles (troposphere) [s].

double tdec\_strat

Life time of particles (stratosphere) [s].

double psc\_h2o

H2O volume mixing ratio for PSC analysis.

double psc\_hno3

HNO3 volume mixing ratio for PSC analysis.

char gw basename [LEN]

Basename for gravity wave variance data.

• char atm\_basename [LEN]

Basename of atmospheric data files.

char atm\_gpfile [LEN]

Gnuplot file for atmospheric data.

double atm\_dt\_out

Time step for atmospheric data output [s].

int atm\_filter

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

· int atm bin

Binary I/O of atmospheric data (0=no, 1=yes).

char csi\_basename [LEN]

Basename of CSI data files.

· double csi\_dt\_out

Time step for CSI data output [s].

• char csi\_obsfile [LEN]

Observation data file for CSI analysis.

• double csi\_obsmin

Minimum observation index to trigger detection.

double csi modmin

Minimum column density to trigger detection [kg/m<sup>2</sup>].

• int csi nz

Number of altitudes of gridded CSI data.

double csi\_z0

Lower altitude of gridded CSI data [km].

· double csi\_z1

Upper altitude of gridded CSI data [km].

int csi\_nx

Number of longitudes of gridded CSI data.

double csi\_lon0

Lower longitude of gridded CSI data [deg].

double csi\_lon1

Upper longitude of gridded CSI data [deg].

· int csi ny

Number of latitudes of gridded CSI data.

• double csi\_lat0

Lower latitude of gridded CSI data [deg].

· double csi\_lat1

Upper latitude of gridded CSI data [deg].

char grid\_basename [LEN]

Basename of grid data files.

char grid\_gpfile [LEN]

Gnuplot file for gridded data.

double grid\_dt\_out

Time step for gridded data output [s].

· int grid\_sparse

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

• int grid\_nz

Number of altitudes of gridded data.

• double grid\_z0

Lower altitude of gridded data [km].

• double grid\_z1

Upper altitude of gridded data [km].

• int grid\_nx

Number of longitudes of gridded data.

• double grid\_lon0

Lower longitude of gridded data [deg].

double grid\_lon1

Upper longitude of gridded data [deg].

• int grid\_ny

Number of latitudes of gridded data.

double grid\_lat0

Lower latitude of gridded data [deg].

• double grid\_lat1

Upper latitude of gridded data [deg].

• char prof\_basename [LEN]

Basename for profile output file.

char prof\_obsfile [LEN]

Observation data file for profile output.

• int prof\_nz

Number of altitudes of gridded profile data.

double prof z0

Lower altitude of gridded profile data [km].

double prof\_z1

Upper altitude of gridded profile data [km].

• int prof\_nx

Number of longitudes of gridded profile data.

· double prof\_lon0

Lower longitude of gridded profile data [deg].

double prof lon1

Upper longitude of gridded profile data [deg].

int prof\_ny

Number of latitudes of gridded profile data.

double prof\_lat0

Lower latitude of gridded profile data [deg].

double prof\_lat1

Upper latitude of gridded profile data [deg].

• char ens\_basename [LEN]

Basename of ensemble data file.

char stat\_basename [LEN]

Basename of station data file.

double stat Ion

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 232 of file libtrac.h.

4.2.2 Field Documentation

4.2.2.1 int ctl\_t::nq

Number of quantities.

Definition at line 235 of file libtrac.h.

4.2.2.2 char ctl\_t::qnt\_name[NQ][LEN]

Quantity names.

Definition at line 238 of file libtrac.h.

4.2.2.3 char ctl\_t::qnt\_unit[NQ][LEN]

Quantity units.

Definition at line 241 of file libtrac.h.

4.2.2.4 char ctl\_t::qnt\_format[NQ][LEN]

Quantity output format.

Definition at line 244 of file libtrac.h.

```
4.2.2.5 int ctl_t::qnt_ens
Quantity array index for ensemble IDs.
Definition at line 247 of file libtrac.h.
4.2.2.6 int ctl_t::qnt_m
Quantity array index for mass.
Definition at line 250 of file libtrac.h.
4.2.2.7 int ctl_t::qnt_rho
Quantity array index for particle density.
Definition at line 253 of file libtrac.h.
4.2.2.8 int ctl_t::qnt_r
Quantity array index for particle radius.
Definition at line 256 of file libtrac.h.
4.2.2.9 int ctl_t::qnt_ps
Quantity array index for surface pressure.
Definition at line 259 of file libtrac.h.
4.2.2.10 int ctl_t::qnt_p
Quantity array index for pressure.
Definition at line 262 of file libtrac.h.
4.2.2.11 int ctl_t::qnt_t
Quantity array index for temperature.
Definition at line 265 of file libtrac.h.
4.2.2.12 int ctl_t::qnt_u
Quantity array index for zonal wind.
Definition at line 268 of file libtrac.h.
4.2.2.13 int ctl_t::qnt_v
```

Quantity array index for meridional wind.

Definition at line 271 of file libtrac.h.

4.2.2.14 int ctl\_t::qnt\_w Quantity array index for vertical velocity. Definition at line 274 of file libtrac.h. 4.2.2.15 int ctl\_t::qnt\_h2o Quantity array index for water vapor vmr. Definition at line 277 of file libtrac.h. 4.2.2.16 int ctl\_t::qnt\_o3 Quantity array index for ozone vmr. Definition at line 280 of file libtrac.h. 4.2.2.17 int ctl\_t::qnt\_theta Quantity array index for potential temperature. Definition at line 283 of file libtrac.h. 4.2.2.18 int ctl\_t::qnt\_pv Quantity array index for potential vorticity. Definition at line 286 of file libtrac.h. 4.2.2.19 int ctl\_t::qnt\_tice Quantity array index for T\_ice. Definition at line 289 of file libtrac.h. 4.2.2.20 int ctl\_t::qnt\_tsts Quantity array index for T\_STS. Definition at line 292 of file libtrac.h. 4.2.2.21 int ctl\_t::qnt\_tnat Quantity array index for T\_NAT. Definition at line 295 of file libtrac.h. 4.2.2.22 int ctl\_t::qnt\_stat Quantity array index for station flag.

Definition at line 298 of file libtrac.h.

```
4.2.2.23 int ctl_t::qnt_gw_u750
Quantity array index for low-level zonal wind.
Definition at line 301 of file libtrac.h.
4.2.2.24 int ctl_t::qnt_gw_v750
Quantity array index for low-level meridional wind.
Definition at line 304 of file libtrac.h.
4.2.2.25 int ctl_t::qnt_gw_sso
Quantity array index for subgrid-scale orography.
Definition at line 307 of file libtrac.h.
4.2.2.26 int ctl_t::qnt_gw_var
Quantity array index for gravity wave variances.
Definition at line 310 of file libtrac.h.
4.2.2.27 int ctl_t::direction
Direction flag (1=forward calculation, -1=backward calculation).
Definition at line 313 of file libtrac.h.
4.2.2.28 double ctl_t::t_start
Start time of simulation [s].
Definition at line 316 of file libtrac.h.
4.2.2.29 double ctl_t::t_stop
Stop time of simulation [s].
Definition at line 319 of file libtrac.h.
4.2.2.30 double ctl_t::dt_mod
Time step of simulation [s].
Definition at line 322 of file libtrac.h.
4.2.2.31 double ctl_t::dt_met
Time step of meteorological data [s].
Definition at line 325 of file libtrac.h.
```

```
4.2.2.32 int ctl_t::met_np
Number of target pressure levels.
Definition at line 328 of file libtrac.h.
4.2.2.33 double ctl_t::met_p[EP]
Target pressure levels [hPa].
Definition at line 331 of file libtrac.h.
4.2.2.34 char ctl_t::met_stage[LEN]
Command to stage meteo data.
Definition at line 334 of file libtrac.h.
4.2.2.35 int ctl_t::isosurf
Isosurface parameter (0=none, 1=pressure, 2=density, 3=theta, 4=balloon).
Definition at line 338 of file libtrac.h.
4.2.2.36 char ctl_t::balloon[LEN]
Balloon position filename.
Definition at line 341 of file libtrac.h.
4.2.2.37 double ctl_t::turb_dx_trop
Horizontal turbulent diffusion coefficient (troposphere) [m^2/s].
Definition at line 344 of file libtrac.h.
4.2.2.38 double ctl_t::turb_dx_strat
Horizontal turbulent diffusion coefficient (stratosphere) [m^2/s].
Definition at line 347 of file libtrac.h.
4.2.2.39 double ctl_t::turb_dz_trop
Vertical turbulent diffusion coefficient (troposphere) [m^2/s].
Definition at line 350 of file libtrac.h.
4.2.2.40 double ctl_t::turb_dz_strat
Vertical turbulent diffusion coefficient (stratosphere) [m<sup>2</sup>/s].
Definition at line 353 of file libtrac.h.
```

```
4.2.2.41 double ctl_t::turb_meso
Scaling factor for mesoscale wind fluctuations.
Definition at line 356 of file libtrac.h.
4.2.2.42 double ctl_t::tdec_trop
Life time of particles (troposphere) [s].
Definition at line 359 of file libtrac.h.
4.2.2.43 double ctl_t::tdec_strat
Life time of particles (stratosphere) [s].
Definition at line 362 of file libtrac.h.
4.2.2.44 double ctl_t::psc_h2o
H2O volume mixing ratio for PSC analysis.
Definition at line 365 of file libtrac.h.
4.2.2.45 double ctl_t::psc_hno3
HNO3 volume mixing ratio for PSC analysis.
Definition at line 368 of file libtrac.h.
4.2.2.46 char ctl_t::gw_basename[LEN]
Basename for gravity wave variance data.
Definition at line 371 of file libtrac.h.
4.2.2.47 char ctl_t::atm_basename[LEN]
Basename of atmospheric data files.
Definition at line 374 of file libtrac.h.
4.2.2.48 char ctl_t::atm_gpfile[LEN]
Gnuplot file for atmospheric data.
Definition at line 377 of file libtrac.h.
4.2.2.49 double ctl_t::atm_dt_out
Time step for atmospheric data output [s].
Definition at line 380 of file libtrac.h.
```

```
4.2.2.50 int ctl_t::atm_filter
Time filter for atmospheric data output (0=no, 1=yes).
Definition at line 383 of file libtrac.h.
4.2.2.51 int ctl_t::atm_bin
Binary I/O of atmospheric data (0=no, 1=yes).
Definition at line 386 of file libtrac.h.
4.2.2.52 char ctl_t::csi_basename[LEN]
Basename of CSI data files.
Definition at line 389 of file libtrac.h.
4.2.2.53 double ctl_t::csi_dt_out
Time step for CSI data output [s].
Definition at line 392 of file libtrac.h.
4.2.2.54 char ctl_t::csi_obsfile[LEN]
Observation data file for CSI analysis.
Definition at line 395 of file libtrac.h.
4.2.2.55 double ctl_t::csi_obsmin
Minimum observation index to trigger detection.
Definition at line 398 of file libtrac.h.
4.2.2.56 double ctl_t::csi_modmin
Minimum column density to trigger detection [kg/m^2].
Definition at line 401 of file libtrac.h.
4.2.2.57 int ctl_t::csi_nz
Number of altitudes of gridded CSI data.
Definition at line 404 of file libtrac.h.
4.2.2.58 double ctl_t::csi_z0
Lower altitude of gridded CSI data [km].
Definition at line 407 of file libtrac.h.
```

```
4.2.2.59 double ctl_t::csi_z1
Upper altitude of gridded CSI data [km].
Definition at line 410 of file libtrac.h.
4.2.2.60 int ctl_t::csi_nx
Number of longitudes of gridded CSI data.
Definition at line 413 of file libtrac.h.
4.2.2.61 double ctl_t::csi_lon0
Lower longitude of gridded CSI data [deg].
Definition at line 416 of file libtrac.h.
4.2.2.62 double ctl_t::csi_lon1
Upper longitude of gridded CSI data [deg].
Definition at line 419 of file libtrac.h.
4.2.2.63 int ctl_t::csi_ny
Number of latitudes of gridded CSI data.
Definition at line 422 of file libtrac.h.
4.2.2.64 double ctl_t::csi_lat0
Lower latitude of gridded CSI data [deg].
Definition at line 425 of file libtrac.h.
4.2.2.65 double ctl_t::csi_lat1
Upper latitude of gridded CSI data [deg].
Definition at line 428 of file libtrac.h.
4.2.2.66 char ctl_t::grid_basename[LEN]
Basename of grid data files.
Definition at line 431 of file libtrac.h.
4.2.2.67 char ctl_t::grid_gpfile[LEN]
Gnuplot file for gridded data.
Definition at line 434 of file libtrac.h.
```

```
4.2.2.68 double ctl_t::grid_dt_out
Time step for gridded data output [s].
Definition at line 437 of file libtrac.h.
4.2.2.69 int ctl_t::grid_sparse
Sparse output in grid data files (0=no, 1=yes).
Definition at line 440 of file libtrac.h.
4.2.2.70 int ctl_t::grid_nz
Number of altitudes of gridded data.
Definition at line 443 of file libtrac.h.
4.2.2.71 double ctl_t::grid_z0
Lower altitude of gridded data [km].
Definition at line 446 of file libtrac.h.
4.2.2.72 double ctl_t::grid_z1
Upper altitude of gridded data [km].
Definition at line 449 of file libtrac.h.
4.2.2.73 int ctl_t::grid_nx
Number of longitudes of gridded data.
Definition at line 452 of file libtrac.h.
4.2.2.74 double ctl_t::grid_lon0
Lower longitude of gridded data [deg].
Definition at line 455 of file libtrac.h.
4.2.2.75 double ctl_t::grid_lon1
Upper longitude of gridded data [deg].
Definition at line 458 of file libtrac.h.
4.2.2.76 int ctl_t::grid_ny
Number of latitudes of gridded data.
```

Definition at line 461 of file libtrac.h.

```
4.2.2.77 double ctl_t::grid_lat0
Lower latitude of gridded data [deg].
Definition at line 464 of file libtrac.h.
4.2.2.78 double ctl_t::grid_lat1
Upper latitude of gridded data [deg].
Definition at line 467 of file libtrac.h.
4.2.2.79 char ctl_t::prof_basename[LEN]
Basename for profile output file.
Definition at line 470 of file libtrac.h.
4.2.2.80 char ctl_t::prof_obsfile[LEN]
Observation data file for profile output.
Definition at line 473 of file libtrac.h.
4.2.2.81 int ctl_t::prof_nz
Number of altitudes of gridded profile data.
Definition at line 476 of file libtrac.h.
4.2.2.82 double ctl_t::prof_z0
Lower altitude of gridded profile data [km].
Definition at line 479 of file libtrac.h.
4.2.2.83 double ctl_t::prof_z1
Upper altitude of gridded profile data [km].
Definition at line 482 of file libtrac.h.
4.2.2.84 int ctl_t::prof_nx
Number of longitudes of gridded profile data.
Definition at line 485 of file libtrac.h.
4.2.2.85 double ctl_t::prof_lon0
Lower longitude of gridded profile data [deg].
Definition at line 488 of file libtrac.h.
```

```
4.2.2.86 double ctl_t::prof_lon1
Upper longitude of gridded profile data [deg].
Definition at line 491 of file libtrac.h.
4.2.2.87 int ctl_t::prof_ny
Number of latitudes of gridded profile data.
Definition at line 494 of file libtrac.h.
4.2.2.88 double ctl_t::prof_lat0
Lower latitude of gridded profile data [deg].
Definition at line 497 of file libtrac.h.
4.2.2.89 double ctl_t::prof_lat1
Upper latitude of gridded profile data [deg].
Definition at line 500 of file libtrac.h.
4.2.2.90 char ctl_t::ens_basename[LEN]
Basename of ensemble data file.
Definition at line 503 of file libtrac.h.
4.2.2.91 char ctl_t::stat_basename[LEN]
Basename of station data file.
Definition at line 506 of file libtrac.h.
4.2.2.92 double ctl_t::stat_lon
Longitude of station [deg].
Definition at line 509 of file libtrac.h.
4.2.2.93 double ctl_t::stat_lat
Latitude of station [deg].
Definition at line 512 of file libtrac.h.
```

```
4.2.2.94 double ctl_t::stat_r
Search radius around station [km].
Definition at line 515 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 552 of file libtrac.h. 4.3.2 Field Documentation 4.3.2.1 double met\_t::time Time [s]. Definition at line 555 of file libtrac.h. 4.3.2.2 int met\_t::nx Number of longitudes. Definition at line 558 of file libtrac.h. 4.3.2.3 int met\_t::ny Number of latitudes. Definition at line 561 of file libtrac.h. 4.3.2.4 int met\_t::np Number of pressure levels. Definition at line 564 of file libtrac.h. 4.3.2.5 double met\_t::lon[EX] Longitude [deg]. Definition at line 567 of file libtrac.h. 4.3.2.6 double met\_t::lat[EY] Latitude [deg]. Definition at line 570 of file libtrac.h. 4.3.2.7 double met\_t::p[EP] Pressure [hPa]. Definition at line 573 of file libtrac.h.

```
4.3.2.8 double met_t::ps[EX][EY]
Surface pressure [hPa].
Definition at line 576 of file libtrac.h.
4.3.2.9 float met_t::pl[EX][EY][EP]
Pressure on model levels [hPa].
Definition at line 579 of file libtrac.h.
4.3.2.10 float met_t::t[EX][EY][EP]
Temperature [K].
Definition at line 582 of file libtrac.h.
4.3.2.11 float met_t::u[EX][EY][EP]
Zonal wind [m/s].
Definition at line 585 of file libtrac.h.
4.3.2.12 float met_t::v[EX][EY][EP]
Meridional wind [m/s].
Definition at line 588 of file libtrac.h.
4.3.2.13 float met_t::w[EX][EY][EP]
Vertical wind [hPa/s].
Definition at line 591 of file libtrac.h.
4.3.2.14 float met_t::h2o[EX][EY][EP]
Water vapor volume mixing ratio [1].
Definition at line 594 of file libtrac.h.
4.3.2.15 float met_t::o3[EX][EY][EP]
Ozone volume mixing ratio [1].
Definition at line 597 of file libtrac.h.
The documentation for this struct was generated from the following file:
```

· libtrac.h

5 File Documentation 23

## 5 File Documentation

#### 5.1 center.c File Reference

Calculate center of mass of air parcels.

#### **Functions**

• int main (int argc, char \*argv[])

## 5.1.1 Detailed Description

Calculate center of mass of air parcels.

Definition in file center.c.

#### 5.1.2 Function Documentation

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

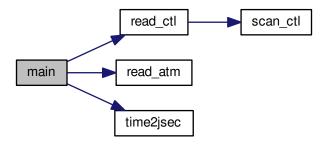
Definition at line 27 of file center.c.

```
00030
00031
        ctl_t ctl;
00032
00033
        atm t *atm;
00034
00035
        FILE *out;
00036
00037
        char tstr[LEN];
00038
00039
        double latm, lats, lonm, lons, t, zm, zs;
00040
00041
        int f, ip, year, mon, day, hour, min;
00042
00043
        /* Allocate... */
00044
        ALLOC(atm, atm_t, 1);
00045
00046
        /* Check arguments... */
00047
        if (argc < 4)
00048
          ERRMSG("Give parameters: <ctl> <outfile> <atm1> [<atm2> ...]");
00049
00050
        /* Read control parameters... */
00051
        read_ctl(argv[1], argc, argv, &ctl);
00052
00053
        /* Write info... */
00054
        printf("Write center of mass data: %s\n", argv[2]);
00055
        /* Create output file... */
if (!(out = fopen(argv[2], "w")))
00056
00057
         ERRMSG("Cannot create file!");
00058
00059
00060
        /* Write header... */
00061
       fprintf(out,
00062
                 "# $1
                       = time [s]\n"
                 "# $2 = altitude (mean) [km]\n"
00063
                 "# $3 = altitude (sigma) [km]\n"
00064
00065
                 "# $4 = altitude (minimum) [km]\n"
00066
                 "# $5 = altitude (10%% percentile) [km]\n"
00067
                 "# $6 = altitude (1st quarter) [km]\n"
                 "# $7 = altitude (median) [km]\n"
00068
                "# $8 = altitude (3rd quarter) [km]\n"
"# $9 = altitude (90%% percentile) [km]\n"
00069
00070
00071
                 "# $10 = altitude (maximum) [km] \n");
00072
       fprintf(out,
```

```
"# $11 = longitude (mean) [deg] n"
00074
                 "# $12 = longitude (sigma) [deg]\n"
                 "# $13 = longitude (minimum) [deg] \n"
00075
                 "# $14 = longitude (10%% percentile) [deg]\n"
00076
                 "# $15 = longitude (1st quarter) [deg] n
00077
00078
                 "# $16 = longitude (median) [deg]\n"
                 "# $17 = longitude (3rd quarter) [deg]\n"
00079
00080
                 "# $18 = longitude (90%% percentile)
00081
                 "# $19 = longitude (maximum) [deg]\n");
        fprintf(out, "# $20 = latitude (mean) [deg]\n
00082
00083
                 "# $21 = latitude (sigma) [deg]\n"
00084
                 "# $22 = latitude (minimum) [deg]\n"
00085
                 "# $23 = latitude (10%% percentile) [deg]\n"
00086
00087
                 "# $24 = latitude (1st quarter) [deg] \n"
                 "# $25 = latitude (median) [deg] n"
00088
                  "# $26 = latitude (3rd quarter) [deg]\n"
00089
                 "# $27 = latitude (90%% percentile) [deg]\n"
00090
                 "# $28 = latitude (maximum) [deg]\n\n");
00091
00092
00093
         /* Loop over files... */
00094
        for (f = 3; f < argc; f++) {</pre>
00095
00096
           /* Read atmopheric data... */
00097
          read_atm(argv[f], &ctl, atm);
00098
00099
           /* Initialize... */
00100
           zm = zs = 0;
           lonm = lons = 0:
00101
00102
          latm = lats = 0:
00103
00104
           /* Calculate mean and standard deviation... */
           for (ip = 0; ip < atm->np; ip++) {
  zm += Z(atm->p[ip]) / atm->np;
00105
00106
             lonm += atm->lon[ip] / atm->np;
latm += atm->lat[ip] / atm->np;
00107
00108
             zs += gsl_pow_2(Z(atm->p[ip])) / atm->np;
00109
             lons += gsl_pow_2(atm->lon[ip]) / atm->np;
00110
00111
             lats += gsl_pow_2(atm->lat[ip]) / atm->np;
00112
00113
          /* Normalize... */
00114
          zs = sqrt(zs - gsl_pow_2(zm));
00115
          lons = sqrt(lons - gsl_pow_2(lonm));
lats = sqrt(lats - gsl_pow_2(latm));
00116
00117
00118
00119
           /* Sort arrays... */
          gsl_sort(atm->p, 1, (size_t) atm->np);
gsl_sort(atm->lon, 1, (size_t) atm->np);
gsl_sort(atm->lat, 1, (size_t) atm->np);
00120
00121
00122
00123
00124
           /* Get time from filename... */
00125
           sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00126
           year = atoi(tstr);
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00127
00128
           mon = atoi(tstr);
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00130
           day = atoi(tstr);
00131
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00132
          hour = atoi(tstr);
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00133
          min = atoi(tstr);
00134
00135
           time2jsec(year, mon, day, hour, min, 0, 0, &t);
00136
           /* Write data... */
00137
00138
           00139
00140
                   Z(atm->p[atm->np - atm->np / 10]),
Z(atm->p[atm->np - atm->np / 4]),
00141
00142
                   Z(atm->p[atm->np / 2]), Z(atm->p[atm->np / 4]), Z(atm->p[atm->np / 10]), Z(atm->p[0]),
00143
00144
                   lonm, lons, atm->lon[0], atm->lon[atm->np / 10],
00145
                   atm->lon[atm->np / 4], atm->lon[atm->np / 2],
atm->lon[atm->np - atm->np / 4],
atm->lon[atm->np - atm->np / 10],
00146
00147
00148
00149
                   atm->lon[atm->np - 1],
                   latm, lats, atm->lat[0], atm->lat[atm->np / 10],
00150
                   atm->lat[atm->np / 4], atm->lat[atm->np / 2], atm->lat[atm->np - atm->np / 4],
00151
00152
                   atm->lat[atm->np - atm->np / 10], atm->lat[atm->np - 1]);
00153
00154
00155
00156
         /* Close file... */
00157
        fclose(out);
00158
00159
        /* Free... */
```

5.2 center.c 25

Here is the call graph for this function:



#### 5.2 center.c

```
00001 /*
        This file is part of MPTRAC.
00002
00003
00004
        MPTRAC is free software: you can redistribute it and/or modify
00005
        it under the terms of the GNU General Public License as published by
00006
        the Free Software Foundation, either version 3 of the License, or
00007
        (at your option) any later version.
00008
00009
        MPTRAC is distributed in the hope that it will be useful,
00010
        but WITHOUT ANY WARRANTY; without even the implied warranty of
00011
        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
        GNU General Public License for more details.
00013
00014
        You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00015
00016
00017
        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
        atm_t *atm;
00034
00035
        FILE *out;
00036
00037
        char tstr[LEN];
00038
00039
        double latm, lats, lonm, lons, t, zm, zs;
00040
00041
        int f, ip, year, mon, day, hour, min;
00042
00043
         /* Allocate... */
00044
        ALLOC(atm, atm_t, 1);
00045
00046
         /* Check arguments... */
00047
          ERRMSG("Give parameters: <ctl> <outfile> <atml> [<atm2> ...]");
00048
00049
00050
        /* Read control parameters... */
00051
        read_ctl(argv[1], argc, argv, &ctl);
00052
```

```
/* Write info... */
00054
        printf("Write center of mass data: %s\n", argv[2]);
00055
00056
        /* Create output file... */
        if (!(out = fopen(argv[2], "w")))
00057
          ERRMSG("Cannot create file!");
00058
00060
         /* Write header... */
00061
        fprintf(out,
00062
                  "# $1
                        = time [s]\n"
                 "# $2 = altitude (mean) [km]\n"
00063
00064
                  "# $3 = altitude (sigma) [km]\n"
00065
                 "# $4 = altitude (minimum) [km] \n"
                 "# $5 = altitude (10%% percentile) [km]\n"
00066
00067
                 "# $6 = altitude (1st quarter) [km]\n"
00068
                 "# $7 = altitude (median) [km]\n"
                 "# $8 = altitude (3rd quarter) [km] n"
00069
                 "# $9 = altitude (90%% percentile) [km]\n"
00070
                 "# $10 = altitude (maximum) [km] n");
00072
        fprintf(out,
                 "# $11 = longitude (mean) [deg]\n"
00073
                 "# $12 = longitude (sigma) [deg]\n"
00074
00075
                 "# $13 = longitude (minimum) [deg] \n"
                 "# $14 = longitude (10%% percentile) [deg]\n"
00076
00077
                 "# $15 = longitude (1st quarter) [deg]\n'
                 "# $16 = longitude (median) [deg]\n"
00078
00079
                 "# $17 = longitude (3rd quarter) [deg]\n"
08000
                 "# $18 = longitude (90%% percentile) [deg]\n"
                 "# $19 = longitude (maximum) [deg]\n");
00081
00082
        fprintf(out,
00083
                  "# $20 = latitude (mean) [deg] n
00084
                 "# $21 = latitude (sigma) [deg]\n"
00085
                 "# $22 = latitude (minimum) [deg] \n"
00086
                 "# $23 = latitude (10%% percentile) [deg] n"
                 "# $24 = latitude (1st quarter) [deg] \n"
00087
                 "# $25 = latitude (median) [deg]\n"
00088
                 "# $26 = latitude (3rd quarter) [deg]\n"
"# $27 = latitude (90%% percentile) [deg]\n"
00089
00091
                 "# $28 = latitude (maximum) [deg] \n\n");
00092
00093
        /* Loop over files... */
00094
        for (f = 3; f < argc; f++) {</pre>
00095
00096
           /* Read atmopheric data... */
00097
          read_atm(argv[f], &ctl, atm);
00098
00099
           /* Initialize... */
00100
          zm = zs = 0;
           lonm = lons = 0;
00101
          latm = lats = 0;
00102
00103
00104
           /\star Calculate mean and standard deviation... \star/
00105
           for (ip = 0; ip < atm->np; ip++) {
            zm += Z(atm->p[ip]) / atm->np;
lonm += atm->lon[ip] / atm->np;
latm += atm->lat[ip] / atm->np;
00106
00107
00108
             zs += gsl_pow_2(Z(atm->p[ip])) / atm->np;
             lons += gsl_pow_2(atm->lon[ip]) / atm->np;
lats += gsl_pow_2(atm->lat[ip]) / atm->np;
00110
00111
00112
00113
          /* Normalize... */
zs = sqrt(zs - gsl_pow_2(zm));
00114
00115
          lons = sqrt(lons - gsl_pow_2(lonm));
lats = sqrt(lats - gsl_pow_2(latm));
00116
00117
00118
00119
           /* Sort arrays... */
          gsl_sort(atm->p, 1, (size_t) atm->np);
gsl_sort(atm->lon, 1, (size_t) atm->np);
gsl_sort(atm->lat, 1, (size_t) atm->np);
00120
00121
00123
00124
           /\star Get time from filename... \star/
           sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00125
           vear = atoi(tstr);
00126
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00127
           mon = atoi(tstr);
00128
00129
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
           day = atoi(tstr);
sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00130
00131
          hour = atoi(tstr);
00132
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00133
00134
           min = atoi(tstr);
00135
           time2jsec(year, mon, day, hour, min, 0, 0, &t);
00136
           00137
00138
00139
```

```
t, zm, zs, Z(atm->p[atm->np-1]),
                                      Z(atm->p[atm->np - atm->np / 10]),
Z(atm->p[atm->np - atm->np / 4]),
00141
00142
                                     Z(atm->p[atm->np - atm->np / 4]),
Z(atm->p[atm->np / 2]), Z(atm->p[atm->np / 4]),
Z(atm->p[atm->np / 10]), Z(atm->p[0]),
lonm, lons, atm->lon[0], atm->lon[atm->np / 10],
atm->lon[atm->np / 4], atm->lon[atm->np / 2],
00143
00144
00145
00146
                                    atm->lon[atm->np / 4], atm->lon[atm->np / 2],
atm->lon[atm->np - atm->np / 4],
atm->lon[atm->np - atm->np / 10],
atm->lon[atm->np - 1],
latm, lats, atm->lat[0], atm->lat[atm->np / 10],
atm->lat[atm->np / 4], atm->lat[atm->np / 2],
atm->lat[atm->np - atm->np / 4],
atm->lat[atm->np - atm->np / 10], atm->lat[atm->np - 1]);
00147
00148
00149
00150
00151
00152
00153
00154
00155
                /* Close file... */
00156
00157
                fclose(out);
00158
                /* Free... */
00160
                free(atm);
00161
00162
                return EXIT_SUCCESS;
00163 }
```

## 5.3 cluster.c File Reference

Clustering of trajectories.

#### **Functions**

• int main (int argc, char \*argv[])

#### 5.3.1 Detailed Description

Clustering of trajectories.

Definition in file cluster.c.

## 5.3.2 Function Documentation

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

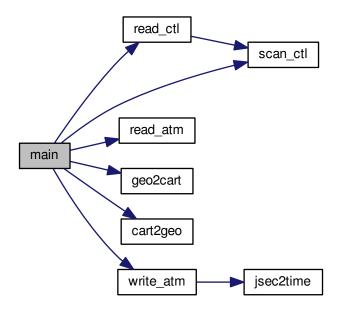
Definition at line 41 of file cluster.c.

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

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

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

Here is the call graph for this function:



### 5.4 cluster.c

```
00001 /*
00002
          This file is part of MPTRAC.
00003
         MPTRAC is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by
00004
00005
00006
          the Free Software Foundation, either version 3 of the License, or
00007
          (at your option) any later version.
00008
00009
          \ensuremath{\mathsf{MPTRAC}} is distributed in the hope that it will be useful,
          but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
00012
          GNU General Public License for more details.
00013
          You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
00016
00017
          Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* --
00028
           Defines...
00029
00030
00032 #define NS 100
00033
00035 #define NT 1000
00036
00037 /* -----
00038
          Main...
00039
00040
00041 int main(
00042 int argc,
00043
         char *argv[]) {
00044
00045
         ctl_t ctl;
00046
```

5.4 cluster.c 31

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

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

5.5 dist.c File Reference 33

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

### 5.5 dist.c File Reference

Calculate transport deviations of trajectories.

### **Functions**

• int main (int argc, char \*argv[])

### 5.5.1 Detailed Description

Calculate transport deviations of trajectories.

Definition in file dist.c.

### 5.5.2 Function Documentation

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

Definition at line 27 of file dist.c.

```
00029
00030
00031
        ctl_t ctl;
00032
00033
        atm_t *atm1, *atm2;
00034
00035
        FILE *out;
00036
00037
        char tstr[LEN];
00038
        double x0[3], x1[3], x2[3], *lon1, *lat1, *p1, *lh1, *lv1, *lon2, *lat2, *p2, *lh2, *lv2, ahtd, avtd, aqtd[NQ], rhtd, rvtd, t;
00039
00040
00041
00042
        int ens, f, ip, iq, np, year, mon, day, hour, min;
00043
00044
        /* Allocate... */
00045
        ALLOC(atm1, atm_t, 1);
00046
        ALLOC(atm2, atm_t, 1);
00047
        ALLOC(lon1, double,
00048
              NP);
00049
        ALLOC(lat1, double,
00050
              NP);
00051
        ALLOC(p1, double,
00052
               NP);
        ALLOC(lh1, double,
00053
00054
              NP);
00055
        ALLOC(lv1, double,
00056
               NP);
00057
        ALLOC(lon2, double,
00058
              NP);
        ALLOC(lat2, double,
00059
00060
              NP);
        ALLOC(p2, double, NP);
00061
00062
00063
        ALLOC(1h2, double,
```

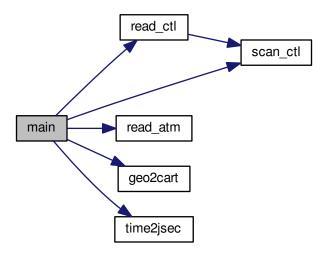
```
00064
               NP);
00065
         ALLOC(1v2, double,
00066
                NP);
00067
00068
         /* Check arguments... */
         if (argc < 5)
00069
00070
           ERRMSG("Give parameters: <ctl> <outfile> <atmla> <atmlb>"
00071
                    " [<atm2a> <atm2b> ...]");
00072
00073
         /* Read control parameters... */
        read_ctl(argv[1], argc, argv, &ctl);
00074
         ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-1", NULL);
00075
00076
00077
         /* Write info... */
00078
         printf("Write transport deviations: sn", argv[2]);
00079
         /* Create output file... */
if (!(out = fopen(argv[2], "w")))
08000
00081
           ERRMSG("Cannot create file!");
00082
00083
00084
         /* Write header... */
         00085
00086
                  "# $2 = AHTD [km]\n"
"# $3 = RHTD [km]\n" "# $4 = AVTD [km]\n" "# $5 = RVTD [km]\n");
00087
00088
         for (iq = 0; iq < ctl.nq; iq++)</pre>
00089
00090
           fprintf(out,
00091
                     "# $%d = AQTD (%s) [%s]\n",
        6 + iq, ctl.qnt_name[iq], ctl.qnt_unit[iq]);
fprintf(out, "\n");
00092
00093
00094
00095
         /* Loop over file pairs... */
00096
         for (f = 3; f < argc; f += 2) {
00097
           /* Read atmopheric data... */
00098
           read_atm(argv[f], &ctl, atm1);
read_atm(argv[f + 1], &ctl, atm2);
00099
00100
00102
           /* Check if structs match... */
00103
           if (atm1->np != atm2->np)
             ERRMSG("Different numbers of parcels!");
00104
           for (ip = 0; ip < atm1->np; ip++)
  if (atm1->time[ip] != atm2->time[ip])
00105
00106
                ERRMSG("Times do not match!");
00107
00108
00109
           /* Init... */
00110
           np = 0;
           ahtd = avtd = rhtd = rvtd = 0;
00111
           for (iq = 0; iq < ctl.nq; iq++)
00112
00113
             aqtd[iq] = 0;
00114
           /* Loop over air parcels... */
for (ip = 0; ip < atm1->np; ip++)
00115
00116
00117
              if (ens < 0 || (ctl.qnt_ens >= 0 && atm1->q[ctl.qnt_ens][ip] == ens)) {
00118
00119
                /* Get Cartesian coordinates... */
                geo2cart(0, atml->lon[ip], atml->lat[ip], x1);
geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00121
00122
00123
                /\star Calculate absolute transport deviations... \star/
                ahtd += DIST(x1, x2);
00124
                avtd += fabs(Z(atm1->p[ip]) - Z(atm2->p[ip]));
00125
                for (iq = 0; iq < ctl.nq; iq++)
   aqtd[iq] += fabs(atml->q[iq][ip] - atm2->q[iq][ip]);
00126
00127
00128
00129
                /\star Calculate relative transport deviations... \star/
                if (f > 3) {
00130
00131
00132
                  /* Get trajectory lengths... */
                  geo2cart(0, lon1[ip], lat1[ip], x0);
lh1[ip] += DIST(x0, x1);
00133
00134
00135
                  lv1[ip] += fabs(Z(p1[ip]) - Z(atm1->p[ip]));
00136
                  geo2cart(0, lon2[ip], lat2[ip], x0);
lh2[ip] += DIST(x0, x2);
lv2[ip] += fabs(Z(p2[ip]) - Z(atm2->p[ip]));
00137
00138
00139
00140
00141
                   /* Get relative transport deviations... */
                  if (lh1[ip] + lh2[ip] > 0)
  rhtd += 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00142
00143
                  if (lv1[ip] + lv2[ip] > 0)

rvtd += 200. * fabs(Z(atml->p[ip]) - Z(atm2->p[ip]))
00144
00145
00146
                       / (lv1[ip] + lv2[ip]);
00147
00148
                 /\star Save positions of air parcels... \star/
00149
                lon1[ip] = atm1->lon[ip];
00150
```

5.5 dist.c File Reference 35

```
00151
               lat1[ip] = atm1->lat[ip];
00152
               p1[ip] = atm1->p[ip];
00153
               lon2[ip] = atm2->lon[ip];
lat2[ip] = atm2->lat[ip];
p2[ip] = atm2->p[ip];
00154
00155
00156
00157
00158
                /\star Increment air parcel counter... \star/
00159
               np++;
00160
00161
           /* Get time from filename... */
00162
           sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00163
00164
           year = atoi(tstr);
00165
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
           mon = atoi(tstr);
sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00166
00167
           day = atoi(tstr);
sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00168
00169
00170
           hour = atoi(tstr);
00171
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00172
           min = atoi(tstr);
00173
           time2jsec(year, mon, day, hour, min, 0, 0, &t);
00174
00175
           /* Write output... */
00176
           fprintf(out, "%.2f %g %g %g %g", t,
                   ahtd / np, rhtd / np, avtd / np, rvtd / np);
00177
           for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");</pre>
00178
00179
             fprintf(out, ctl.qnt_format[iq], aqtd[iq] / np);
00180
00181
00182
           fprintf(out, "\n");
00183
00184
00185
         /* Close file... */
00186
        fclose(out);
00187
00188
        /* Free... */
00189
        free(atm1);
00190
        free(atm2);
00191
         free(lon1);
00192
        free(lat1);
        free(p1);
00193
00194
        free(lh1);
00195
        free(lv1);
00196
        free(lon2);
00197
        free(lat2);
00198
        free(p2);
00199
        free(lh2);
00200
        free(1v2);
00201
00202
        return EXIT_SUCCESS;
00203 }
```

Here is the call graph for this function:



## 5.6 dist.c

```
00001 /*
00002
         This file is part of MPTRAC.
00003
00004
         MPTRAC is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by
00005
00006
         the Free Software Foundation, either version 3 of the License, or
00007
         (at your option) any later version.
00008
         MPTRAC is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
00009
00010
00011
         MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
         GNU General Public License for more details.
00013
         You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
00016
00017
         Copright (C) 2013-2018 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028
         int argc,
         char *argv[]) {
00029
00030
00031
         ctl_t ctl;
00032
00033
         atm_t *atm1, *atm2;
00034
00035
         FILE *out;
00036
00037
         char tstr[LEN];
00038
         double x0[3], x1[3], x2[3], \starlon1, \starlat1, \starp1, \starlh1, \starlv1,
00039
00040
           *lon2, *lat2, *p2, *lh2, *lv2, ahtd, avtd, aqtd[NQ], rhtd, rvtd, t;
00041
00042
         int ens, f, ip, iq, np, year, mon, day, hour, min;
00043
00044
          /* Allocate... */
00045
         ALLOC(atm1, atm_t, 1);
00046
         ALLOC(atm2, atm_t, 1);
         ALLOC(lon1, double,
00047
00048
                NP);
         ALLOC(lat1, double,
```

5.6 dist.c 37

```
NP);
00051
        ALLOC(p1, double,
00052
              NP);
        ALLOC(lh1, double,
00053
00054
              NP);
00055
        ALLOC(lv1, double,
               NP);
00057
        ALLOC(lon2, double,
00058
              NP);
00059
        ALLOC(lat2, double,
00060
              NP);
        ALLOC(p2, double,
00061
00062
               NP);
00063
        ALLOC(1h2, double,
00064
              NP);
00065
        ALLOC(1v2, double,
00066
              NP);
00067
00068
        /* Check arguments... */
00069
        if (argc < 5)
00070
         ERRMSG("Give parameters: <ctl> <outfile> <atmla> <atmlb>"
                  " [<atm2a> <atm2b> ...]");
00071
00072
00073
        /* Read control parameters... */
        read_ctl(argv[1], argc, argv, &ctl);
00074
00075
        ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-1", NULL);
00076
        /* Write info... */
00077
00078
        printf("Write transport deviations: %s\n", argv[2]);
00079
        /* Create output file... */
if (!(out = fopen(argv[2], "w")))
00080
00081
00082
          ERRMSG("Cannot create file!");
00083
00084
        /* Write header... */
        00085
00086
                 "# $2 = AHTD [km] \n"
00088
                "# $3 = RHTD [km]\n" "# $4 = AVTD [km]\n" "# $5 = RVTD [km]\n");
00089
        for (iq = 0; iq < ctl.nq; iq++)</pre>
          00090
00091
        6 + iq, ctl.qnt_name[iq], ctl.qnt_unit[iq]);
fprintf(out, "\n");
00092
00093
00094
00095
        /* Loop over file pairs... */
00096
        for (f = 3; f < argc; f += 2) {</pre>
00097
00098
          /* Read atmopheric data... */
          read_atm(argv[f], &ctl, atm1);
read_atm(argv[f + 1], &ctl, atm2);
00099
00100
00101
00102
          /* Check if structs match... */
00103
          if (atm1->np != atm2->np)
            ERRMSG("Different numbers of parcels!");
00104
          for (ip = 0; ip < atm1->np; ip++)
  if (atm1->time[ip] != atm2->time[ip])
00105
00107
              ERRMSG("Times do not match!");
00108
          /* Init... */
00109
00110
          np = 0:
          ahtd = avtd = rhtd = rvtd = 0;
00111
          for (iq = 0; iq < ctl.nq; iq++)</pre>
00112
00113
            aqtd[iq] = 0;
00114
00115
          /* Loop over air parcels... */
          for (ip = 0; ip < atml->np; ip++)
  if (ens < 0 || (ctl.qnt_ens >= 0 && atml->q[ctl.qnt_ens][ip] == ens)) {
00116
00117
00118
00119
               /* Get Cartesian coordinates... */
00120
               geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00121
               geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00122
               /\star Calculate absolute transport deviations... \star/
00123
               ahtd += DIST(x1, x2);
00124
00125
               avtd += fabs(Z(atm1->p[ip]) - Z(atm2->p[ip]));
00126
               for (iq = 0; iq < ctl.nq; iq++)</pre>
00127
                aqtd[iq] += fabs(atm1->q[iq][ip] - atm2->q[iq][ip]);
00128
00129
               /* Calculate relative transport deviations... */
               if (f > 3) {
00130
00131
00132
                 /* Get trajectory lengths... */
00133
                 geo2cart(0, lon1[ip], lat1[ip], x0);
                 lh1[ip] += DIST(x0, x1);
00134
                 lv1[ip] += fabs(Z(p1[ip]) - Z(atml->p[ip]));
00135
00136
```

```
geo2cart(0, lon2[ip], lat2[ip], x0);
00138
                  lh2[ip] += DIST(x0, x2);
                 lv2[ip] += fabs(Z(p2[ip]) - Z(atm2->p[ip]));
00139
00140
00141
                 /* Get relative transport deviations... */
                 if (lh1[ip] + lh2[ip] > 0)
rhtd += 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00142
00143
00144
                  if (lv1[ip] + lv2[ip] > 0)
                  rvtd += 200. * fabs(Z(atm1->p[ip]) - Z(atm2->p[ip]))
/ (lv1[ip] + lv2[ip]);
00145
00146
00147
00148
               /\star Save positions of air parcels... \star/
00149
               lon1[ip] = atml->lon[ip];
lat1[ip] = atml->lat[ip];
00150
00151
00152
               p1[ip] = atm1->p[ip];
00153
               lon2[ip] = atm2->lon[ip];
lat2[ip] = atm2->lat[ip];
00154
00155
00156
               p2[ip] = atm2->p[ip];
00157
00158
                /\star Increment air parcel counter... \star/
00159
               np++;
00160
00161
           /* Get time from filename... */
00162
00163
           sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
           year = atoi(tstr);
sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00164
00165
00166
           mon = atoi(tstr);
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00167
00168
           day = atoi(tstr);
00169
           sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00170
           hour = atoi(tstr);
00171
           sprintf(tstr, \ \ \ \ \ \ \ \&argv[f][strlen(argv[f]) \ - \ \ 6]);
           min = atoi(tstr);
00172
00173
           time2jsec(year, mon, day, hour, min, 0, 0, &t);
00174
00175
           /* Write output... */
           00176
00177
           for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl.qnt_format[iq], aqtd[iq] / np);</pre>
00178
00179
00180
00181
00182
           fprintf(out, "\n");
00183
00184
         /* Close file... */
00185
00186
        fclose(out);
00187
00188
         /* Free... */
00189
         free(atm1);
00190
        free(atm2);
00191
         free(lon1):
00192
         free(lat1);
         free(p1);
00194
         free(lh1);
00195
         free(lv1);
00196
        free (lon2);
00197
        free (lat2);
00198
        free (p2);
00199
         free(lh2);
00200
        free(lv2);
00201
00202
        return EXIT_SUCCESS;
00203 }
```

## 5.7 extract.c File Reference

Extract single trajectory from atmospheric data files.

## **Functions**

• int main (int argc, char \*argv[])

## 5.7.1 Detailed Description

Extract single trajectory from atmospheric data files.

Definition in file extract.c.

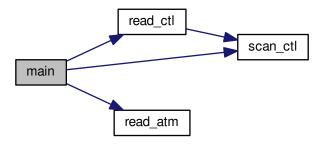
## 5.7.2 Function Documentation

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

Definition at line 27 of file extract.c.

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

Here is the call graph for this function:



## 5.8 extract.c

```
00001 /*
00002
        This file is part of MPTRAC.
00003
00004
        MPTRAC is free software: you can redistribute it and/or modify
00005
        it under the terms of the GNU General Public License as published by
00006
        the Free Software Foundation, either version 3 of the License, or
00007
        (at your option) any later version.
00008
00009
        MPTRAC is distributed in the hope that it will be useful,
00010
        but WITHOUT ANY WARRANTY; without even the implied warranty of
00011
        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
        GNU General Public License for more details.
00013
        You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
00016
00017
        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
        atm_t *atm;
00034
00035
        FILE *in, *out;
00036
00037
        int f, ip, iq;
00038
00039
         /* Allocate... */
00040
        ALLOC(atm, atm_t, 1);
00041
00042
         /* Check arguments... */
00043
        if (argc < 4)
00044
          ERRMSG("Give parameters: <ctl> <outfile> <atm1> [<atm2> ...]");
00045
00046
        /* Read control parameters... */
00047
         read_ctl(argv[1], argc, argv, &ctl);
00048
        ip = (int) scan_ctl(argv[1], argc, argv, "EXTRACT_IP", -1, "0", NULL);
00049
        /* Write info... */ printf("Write trajectory data: s^n, argv[2]);
00050
00051
00052
00053
         /* Create output file... */
        if (!(out = fopen(argv[2], "w")))
00054
          ERRMSG("Cannot create file!");
00055
00056
00057
         /* Write header... */
00058
        fprintf(out,
00059
                 "# $1 = time [s] \n"
```

5.9 init.c File Reference 41

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

fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00062
00063
00064
        ctl.qnt_unit[iq]);
fprintf(out, "\n");
00065
00066
00067
        /* Loop over files... */
00068
        for (f = 3; f < argc; f++) {</pre>
00069
          /* Read atmopheric data... */
if (!(in = fopen(argv[f], "r")))
00070
00071
00072
             continue;
00073
00074
             fclose(in);
00075
           read_atm(argv[f], &ctl, atm);
00076
00077
          /* Write data... */
fprintf(out, "%.2f %g %g %g", atm->time[ip],
00078
                    Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
00079
           for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");</pre>
08000
00081
00082
             fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00083
00084
           fprintf(out, "\n");
00085 }
00086
00087
        /* Close file... */
00088
        fclose(out);
00089
00090
        /* Free... */
00091
        free(atm);
00092
00093
        return EXIT_SUCCESS;
00094 }
```

## 5.9 init.c File Reference

Create atmospheric data file with initial air parcel positions.

#### **Functions**

• int main (int argc, char \*argv[])

#### 5.9.1 Detailed Description

Create atmospheric data file with initial air parcel positions.

Definition in file init.c.

## 5.9.2 Function Documentation

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

Definition at line 27 of file init.c.

```
00029
00030
00031
               atm_t *atm;
00032
00033
               ctl t ctl;
00034
               gsl_rng *rng;
00036
00037
               double dt, dz, dlon, dlat, lat0, lat1, lon0, lon1, t0, t1, z0, z1,
00038
                  t, z, lon, lat, st, sz, slon, slat, sx, ut, uz, ulon, ulat, m;
00039
00040
              int even, ip, irep, rep;
00041
00042
                /* Allocate... */
00043
               ALLOC(atm, atm_t, 1);
00044
00045
               /* Check arguments... */
00046
               if (argc < 3)
00047
                   ERRMSG("Give parameters: <ctl> <atm_out>");
00048
00049
               /* Read control parameters... */
              read_ctl(argv[1], argc, argv, &ctl);

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

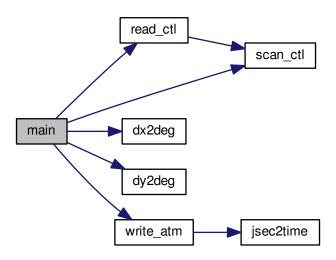
t1 = scan_ctl(argv[1], argc, argv, "INIT_T1", -1, "0", NULL);

dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
00050
00051
00052
             t1 = scan_ctl(argv[1], argc, argv, "INIT_TI", -1, "0", NULL);
dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
z0 = scan_ctl(argv[1], argc, argv, "INIT_ZO", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "INIT_Z1", -1, "0", NULL);
dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);
lon0 = scan_ctl(argv[1], argc, argv, "INIT_LONO", -1, "0", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "INIT_LONO", -1, "0", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "INIT_DLON", -1, "1", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "INIT_LATO", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT1", -1, "0", NULL);
dlat = scan_ctl(argv[1], argc, argv, "INIT_DLAT", -1, "1", NULL);
st = scan_ctl(argv[1], argc, argv, "INIT_SINT, -1, "0", NULL);
sz = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
slat = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
sx = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_SX", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_UZ", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_UZ", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
ut = scan_ctl(argv[1], argc, argv, "INIT_EVENLY", -1, "1", NULL);
rep = (int) scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
00053
00054
00055
00056
00057
00058
00059
00060
00061
00062
00063
00064
00065
00067
00068
00069
00070
00071
00072
00073
00074
00075
00076
              /* Initialize random number generator... */
00077
               gsl_rng_env_setup();
00078
               rng = gsl_rng_alloc(gsl_rng_default);
00080
               /* Create grid... */
00081
               for (t = t0; t <= t1; t += dt)</pre>
                  for (z = z0; z <= z1; z += dz)
for (lon = lon0; lon <= lon1; lon += dlon)
00082
00083
                           for (lat = lat0; lat <= lat1; lat += dlat)
  for (irep = 0; irep < rep; irep++) {</pre>
00084
00086
00087
                                   /* Set position... */
00088
                                   atm->time[atm->np]
00089
                                      = (t + gsl_ran_gaussian_ziggurat(rng, st / 2.3548)
00090
                                           + ut * (gsl_rng_uniform(rng) - 0.5));
00091
                                   atm->p[atm->np]
00092
                                     = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
00093
                                               + uz * (gsl_rng_uniform(rng) - 0.5));
00094
                                   atm->lon[atm->np]
                                       = (lon + gsl_ran_gaussian_ziggurat(rng, slon / 2.3548)
00095
                                            + gsl_ran_gaussian_ziggurat(rng, dx2deg(sx, lat) / 2.3548)
+ ulon * (gsl_rng_uniform(rng) - 0.5));
00096
00097
00098
                                  do {
00099
                                      atm->lat[atm->np]
00100
                                           = (lat + gsl_ran_gaussian_ziggurat(rng, slat / 2.3548)
                                                + gsl_ran_gaussian_ziggurat(rng, dy2deg(sx) / 2.3548)
+ ulat * (gsl_rng_uniform(rng) - 0.5));
00101
00102
                                   } while (even && gsl_rng_uniform(rng) >
00103
                                                    fabs(cos(atm->lat[atm->np] * M_PI / 180.)));
00104
00105
00106
                                   /* Set particle counter... */
00107
                                   if ((++atm->np) >= NP)
                                       ERRMSG("Too many particles!");
00108
00109
00110
00111
               /* Check number of air parcels... */
00112
               if (atm->np <= 0)
00113
                  ERRMSG("Did not create any air parcels!");
00114
00115
               /* Initialize mass... */
```

5.10 init.c 43

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

Here is the call graph for this function:



# 5.10 init.c

```
00001 /*
00002
        This file is part of MPTRAC.
00004
        MPTRAC is free software: you can redistribute it and/or modify
00005
        it under the terms of the GNU General Public License as published by
00006
        the Free Software Foundation, either version 3 of the License, or
00007
        (at your option) any later version.
80000
00009
        MPTRAC is distributed in the hope that it will be useful,
00010
        but WITHOUT ANY WARRANTY; without even the implied warranty of
00011
        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
        GNU General Public License for more details.
00013
        You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
00016
00017
        Copright (C) 2013-2018 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028
        int argc,
00029
        char *argv[]) {
00030
00031
        atm t *atm;
00032
00033
        ctl_t ctl;
```

```
00034
00035
                qsl rnq *rnq;
00036
00037
               double dt, dz, dlon, dlat, lat0, lat1, lon0, lon1, t0, t1, z0, z1, \,
00038
                   t, z, lon, lat, st, sz, slon, slat, sx, ut, uz, ulon, ulat, m;
00039
00040
                int even, ip, irep, rep;
00041
                 /* Allocate... */
00042
00043
               ALLOC(atm, atm_t, 1);
00044
00045
                /* Check arguments... */
                if (argc < 3)
00046
00047
                    ERRMSG("Give parameters: <ctl> <atm_out>");
00048
00049
                /* Read control parameters... */
               read_ctl(argv[1], argc, argv, &ctl);
t0 = scan_ctl(argv[1], argc, argv, "INIT_TO", -1, "0", NULL);
t1 = scan_ctl(argv[1], argc, argv, "INIT_T1", -1, "0", NULL);
00050
00051
              tl = scan_ctl(argv[1], argc, argv, "INIT_TO", -1, "0", NULL);

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

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

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

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

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

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

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

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

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

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

st = scan_ctl(argv[1], argc, argv, "INIT_DLAT", -1, "0", NULL);

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

ut = sca
00053
00054
00055
00056
00057
00058
00059
00060
00061
00062
00063
00064
00065
00066
00067
00068
00069
00070
                even = (int) scan_ctl(argy[1], argc, argv, "INIT_EVENLY", -1, "1", NU rep = (int) scan_ctl(argv[1], argc, argv, "INIT_REP", -1, "1", NULL); m = scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
00072
00073
00074
00075
00076
               /* Initialize random number generator... */
00077
               gsl_rng_env_setup();
00078
               rng = gsl_rng_alloc(gsl_rng_default);
00079
08000
                 /* Create grid... */
00081
                for (t = t0; t <= t1; t += dt)</pre>
                    for (z = z0; z \le z1; z += dz)
00082
                        for (lon = lon0; lon <= lon1; lon += dlon)</pre>
00083
                             for (lat = lat0; lat <= lat1; lat += dlat)</pre>
00085
                                 for (irep = 0; irep < rep; irep++) {</pre>
00086
00087
                                      /* Set position... */
00088
                                     atm->time[atm->np]
00089
                                         = (t + gsl_ran_gaussian_ziggurat(rng, st / 2.3548)
                                               + ut * (gsl_rng_uniform(rng) - 0.5));
00091
                                     atm->p[atm->np]
00092
                                         = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
00093
                                                 + uz * (gsl_rng_uniform(rng) - 0.5));
                                     atm->lon[atm->np]
00094
00095
                                         = (lon + gsl_ran_gaussian_ziggurat(rng, slon / 2.3548)
                                               + gsl_ran_gaussian_ziggurat(rng, dx2deg(sx, lat) / 2.3548)
+ ulon * (gsl_rng_uniform(rng) - 0.5));
00096
00097
00098
                                     do {
00099
                                         atm->lat[atm->np]
                                              = (lat + gsl_ran_gaussian_ziggurat(rng, slat / 2.3548)
00100
00101
                                                    + gsl_ran_gaussian_ziggurat(rng, dy2deg(sx) / 2.3548)
+ ulat * (gsl_rng_uniform(rng) - 0.5));
00102
                                     } while (even && gsl_rng_uniform(rng) >
00104
                                                        fabs(cos(atm->lat[atm->np] * M_PI / 180.)));
00105
                                     /* Set particle counter... */
if ((++atm->np) >= NP)
00106
00107
                                         ERRMSG("Too many particles!");
00108
00109
00110
00111
                 /* Check number of air parcels... */
00112
                if (atm->np <= 0)
                    ERRMSG("Did not create any air parcels!");
00113
00114
                /* Initialize mass... */
00116
                if (ctl.qnt_m >= 0)
00117
                    for (ip = 0; ip < atm->np; ip++)
00118
                        atm->q[ctl.qnt_m][ip] = m / atm->np;
00119
00120
               /* Save data... */
```

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

# 5.11 jsec2time.c File Reference

Convert Julian seconds to date.

#### **Functions**

• int main (int argc, char \*argv[])

## 5.11.1 Detailed Description

Convert Julian seconds to date.

Definition in file jsec2time.c.

## 5.11.2 Function Documentation

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

Definition at line 27 of file jsec2time.c.

```
00029
00030
00031
         double jsec, remain;
00032
00033
         int day, hour, min, mon, sec, year;
00034
00035
          /\!\star Check arguments... \star/
          if (argc < 2)
    ERRMSG("Give parameters: <jsec>");
00036
00037
00038
00039
          /* Read arguments... */
00040
         jsec = atof(argv[1]);
00041
00042
         /* Convert time... */
         jsec2time(jsec, &year, &mon, &day, &hour, &min, &sec, &remain);
printf("%d %d %d %d %d %d %g\n", year, mon, day, hour, min, sec, remain);
00043
00044
00045
00046
          return EXIT_SUCCESS;
```

Here is the call graph for this function:



# 5.12 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
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/</a>.
00016
00017
        Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028 int arg
        int argc,
00029
        char *argv[]) {
00030
        double jsec, remain;
00032
00033
        int day, hour, min, mon, sec, year;
00034
00035
        /* Check arguments... */
00036
        if (argc < 2)
00037
          ERRMSG("Give parameters: <jsec>");
00038
00039
         /* Read arguments.
00040
        jsec = atof(argv[1]);
00041
00042
        /* Convert time... */
00043
         jsec2time(jsec, &year, &mon, &day, &hour, &min, &sec, &remain);
00044
        printf("%d %d %d %d %d %g\n", year, mon, day, hour, min, sec, remain);
00045
00046
        return EXIT_SUCCESS;
00047 }
```

#### 5.13 libtrac.c File Reference

MPTRAC library definitions.

#### **Functions**

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

Convert Cartesian coordinates to geolocation.

• double deg2dx (double dlon, double lat)

Convert degrees to horizontal distance.

double deg2dy (double dlat)

Convert degrees to horizontal distance.

double dp2dz (double dp, double p)

Convert pressure to vertical distance.

double dx2deg (double dx, double lat)

Convert horizontal distance to degrees.

double dy2deg (double dy)

Convert horizontal distance to degrees.

double dz2dp (double dz, double p)

Convert vertical distance to pressure.

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

Convert geolocation to Cartesian coordinates.

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

Get meteorological data for given timestep.

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

Get meteorological data for timestep.

void intpol met 2d (double array[EX][EY], int ix, int iy, double wx, double wy, double \*var)

Linear interpolation of 2-D meteorological data.

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

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

Spatial interpolation of meteorological data.

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

Temporal interpolation of meteorological data.

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

Convert seconds to date.

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

Find array index.

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

Read atmospheric data.

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

Read control parameters.

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

Read meteorological data file.

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

Extrapolate meteorological data at lower boundary.

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

Read and convert variable from meteorological data file.

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

Convert meteorological data from model levels to pressure levels.

void read met periodic (met t \*met)

Create meteorological data with periodic boundary conditions.

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

Read a control parameter from file or command line.

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

Convert date to seconds.

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

Measure wall-clock time.

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

Write atmospheric data.

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

Write CSI data.

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

Write ensemble data.

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

Write gridded data.

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

Write profile data.

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

Write station data.

## 5.13.1 Detailed Description

MPTRAC library definitions.

Definition in file libtrac.c.

## 5.13.2 Function Documentation

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

Convert Cartesian coordinates to geolocation.

Definition at line 29 of file libtrac.c.

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

5.13.2.2 double deg2dx ( double dlon, double lat )

Convert degrees to horizontal distance.

Definition at line 45 of file libtrac.c.

5.13.2.3 double deg2dy ( double dlat )

Convert degrees to horizontal distance.

Definition at line 54 of file libtrac.c.

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

5.13.2.4 double dp2dz ( double dp, double p )

Convert pressure to vertical distance.

Definition at line 62 of file libtrac.c.

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

## 5.13.2.5 double dx2deg ( double dx, double lat )

Convert horizontal distance to degrees.

Definition at line 71 of file libtrac.c.

## 5.13.2.6 double dy2deg ( double dy )

Convert horizontal distance to degrees.

Definition at line 84 of file libtrac.c.

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

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

Convert vertical distance to pressure.

Definition at line 92 of file libtrac.c.

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

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

Convert geolocation to Cartesian coordinates.

Definition at line 101 of file libtrac.c.

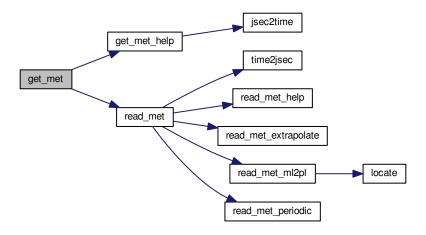
5.13.2.9 void get\_met ( ctl\_t \* ctl, char \* metbase, double t, met\_t \* met0, met\_t \* met1 )

Get meteorological data for given timestep.

Definition at line 117 of file libtrac.c.

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

Here is the call graph for this function:



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

Get meteorological data for timestep.

Definition at line 156 of file libtrac.c.

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

Here is the call graph for this function:



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

Linear interpolation of 2-D meteorological data.

Definition at line 182 of file libtrac.c.

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

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

Linear interpolation of 3-D meteorological data.

Definition at line 206 of file libtrac.c.

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

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

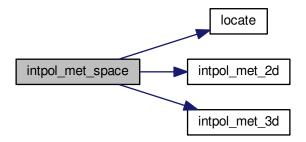
5.13.2.13 void intpol\_met\_space ( met\_t \* met, double p, double lon, double lon

Spatial interpolation of meteorological data.

Definition at line 236 of file libtrac.c.

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

Here is the call graph for this function:



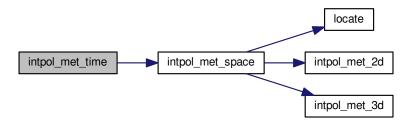
5.13.2.14 void intpol\_met\_time ( met\_t \* met0, met\_t \* met1, double ts, double p, double lon, double lat, double \* ps, double \* t, double \* u, double \* v, double \* w, double \* b2o, double \* o3 )

Temporal interpolation of meteorological data.

Definition at line 286 of file libtrac.c.

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

Here is the call graph for this function:



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

Convert seconds to date.

Definition at line 341 of file libtrac.c.

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

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

Find array index.

Definition at line 376 of file libtrac.c.

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

```
ihi = i;
00392
00393
                ilo = i;
00394
         } else
           while (ihi > ilo + 1) {
  i = (ihi + ilo) >> 1;
00395
00396
              if (xx[i] <= x)</pre>
00398
                ihi = i;
00399
              else
00400
                ilo = i;
           }
00401
00402
00403
         return ilo;
00404 }
```

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

Read atmospheric data.

Definition at line 408 of file libtrac.c.

```
00411
00412
00413
          FILE *in;
00414
00415
         char line[LEN], *tok;
00416
00417
          int iq;
00418
          /* Init... */
00420
          atm->np = 0;
00421
         /* Write info... */
printf("Read atmospheric data: %s\n", filename);
00422
00423
00424
00425
          /* Open file... *,
         if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");
00426
00427
00428
          /* Read binary data... */
if (ctl->atm_bin && ctl->atm_gpfile[0] == '-') {
00429
00430
00431
            FREAD(&atm->np, int,
00432
00433
                    in);
            FREAD(atm->time, double,
00434
00435
                      (size_t) atm->np,
00436
                    in);
00437
            FREAD(atm->p, double,
00438
                       (size_t) atm->np,
00439
                    in);
            FREAD(atm->lon, double,
00440
00441
                      (size_t) atm->np,
00442
                    in);
00443
            FREAD(atm->lat, double,
00444
                       (size_t) atm->np,
00445
                    in);
00446
            for (iq = 0; iq < ctl->nq; iq++)
00447
              FREAD(atm->q[iq], double,
                          (size_t) atm->np,
00448
00449
                       in);
00450
00451
00452
          /* Read ASCII data... */
00453
         else {
00454
00455
             /* Read line... */
00456
            while (fgets(line, LEN, in)) {
00457
               /* 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]);
00458
00459
00460
00461
00462
00463
00464
00465
00466
               /\star Convert altitude to pressure... \star/
               atm->p[atm->np] = P(atm->p[atm->np]);
00467
00468
00469
               /* Increment data point counter... */
```

```
if ((++atm->np) > NP)
              ERRMSG("Too many data points!");
00471
00472
          }
        }
00473
00474
00475
        /* Close file... */
00476
        fclose(in);
00477
00478
         /* Check number of points... */
        if (atm->np < 1)
   ERRMSG("Can not read any data!");</pre>
00479
00480
00481 }
```

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

Read control parameters.

Definition at line 485 of file libtrac.c.

```
00489
00490
00491
       int ip, iq;
00492
00493
        /* Write info... */
       00494
00495
               argv[0], __DATE__, __TIME__);
00496
00497
00498
       /* Initialize quantity indices... */
       ctl->qnt_ens = -1;
ctl->qnt_m = -1;
ctl->qnt_r = -1;
00500
00501
       ctl->qnt_rho = -1;
ctl->qnt_ps = -1;
00502
00503
       ctl->qnt_p = -1;
00504
00505
       ctl->qnt_t = -1;
00506
       ctl->qnt_u = -1;
00507
        ctl->qnt_v = -1;
       ctl->qnt_w = -1;
00508
        ct1->qnt_h2o = -1;
00509
00510
        ct1->qnt_o3 = -1;
        ctl->qnt_theta = -1;
00511
00512
        ctl \rightarrow qnt_pv = -1;
00513
        ctl->qnt\_tice = -1;
       ctl->qnt_tsts = -1;
00514
00515
       ctl->qnt_tnat = -1;
00516
       ctl->qnt_gw_var = -1;
00517
       ctl->qnt_stat = -1;
00518
00519
        /* \ {\tt Read \ quantities...} \ */
        \verb|ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);|
00520
       if (ctl->nq > NQ)
   ERRMSG("Too many quantities!");
00521
00522
00523
        for (iq = 0; iq < ctl->nq; iq++) {
00524
          00525
00526
00527
00528
00529
          /* Try to identify quantity... */
if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
00531
00532
            ctl->qnt_ens = iq;
           sprintf(ctl->qnt_unit[iq], "-");
00533
          } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
  ctl->qnt_m = iq;
00534
00535
            sprintf(ctl->qnt_unit[iq], "kg");
00536
         00537
00538
            sprintf(ctl->qnt_unit[iq], "m");
00539
         } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
  ctl->qnt_rho = iq;
  sprintf(ctl->qnt_unit[iq], "kg/m^3");
00540
00541
00543
         } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
00544
           ctl->qnt_ps = iq;
           sprintf(ctl->qnt_unit[iq], "hPa");
00545
          } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
ctl->qnt_p = iq;
00546
00547
00548
            sprintf(ctl->qnt_unit[iq], "hPa");
          } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
```

```
ctl->qnt_t = iq;
             sprintf(ctl->qnt_unit[iq], "K");
00551
00552
           } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
             ctl->qnt_u = iq;
00553
             sprintf(ctl->qnt_unit[iq], "m/s");
00554
           } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
  ctl->qnt_v = iq;
00555
00557
             sprintf(ctl->qnt_unit[iq], "m/s");
00558
           } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
             ctl->qnt_w = iq;
00559
             sprintf(ctl->qnt_unit[iq], "hPa/s");
00560
           } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
  ctl->qnt_h2o = iq;
00561
00562
00563
             sprintf(ctl->qnt_unit[iq], "1");
00564
           } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
             ctl->qnt_o3 = iq;
00565
             sprintf(ctl->qnt_unit[iq], "1");
00566
           } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
  ctl->qnt_theta = iq;
00567
00568
             sprintf(ctl->qnt_unit[iq], "K");
00569
00570
           } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
             ctl->qnt_pv = iq;
00571
             sprintf(ctl->qnt_unit[iq], "PVU");
00572
           } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
00573
             ctl->qnt_tice = iq;
sprintf(ctl->qnt_unit[iq], "K");
00574
00575
00576
           } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
00577
             ctl->qnt_tsts = iq;
00578
             sprintf(ctl->qnt_unit[iq], "K");
           } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
  ctl->qnt_tnat = iq;
00579
00580
00581
             sprintf(ctl->qnt_unit[iq], "K");
00582
           } else if (strcmp(ctl->qnt_name[iq], "gw_var") == 0) {
00583
             ctl->qnt_gw_var = iq;
             sprintf(ctl->qnt_unit[iq], "K^2");
00584
           } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
ctl->qnt_stat = iq;
00585
00586
             sprintf(ctl->qnt_unit[iq], "-");
00588
           } else
00589
             scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00590
00591
00592
         /* Time steps of simulation... */
00593
         ctl->direction =
           (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
f (ctl->direction != -1 && ctl->direction != 1)
00594
00595
00596
          ERRMSG("Set DIRECTION to -1 or 1!");
00597
         ctl->t start
00598
           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);
00599
00600
00601
00602
         /* Meteorological data... */
         ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
00603
00604
00605
         if (ctl->met np > EP)
           ERRMSG("Too many levels!");
00606
00607
         for (ip = 0; ip < ctl->met_np; ip++)
         ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
00608
00609
00610
00611
         /* Isosurface parameters... */
00612
         ctl->isosurf
         = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00613
00614
00615
00616
        /* Diffusion parameters... */
00617
        ctl->turb dx trop
00618
           = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00619
        ctl->turb_dx_strat
00620
           = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
         ctl->turb_dz_trop
00621
00622
           = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00623
         ctl->turb dz strat
            = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00624
00625
         ctl->turb_meso =
00626
           scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00627
00628
        /* Life time of particles... */
         ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
00629
         ctl->tdec_strat =
00630
00631
           scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00632
00633
         /* PSC analysis... ∗/
00634
         ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
         ctl->psc_hno3 =
00635
00636
           scan ctl(filename, argc, argv, "PSC HNO3", -1, "9e-9", NULL);
```

```
00637
00638
         /* Gravity wave analysis... */
         scan_ctl(filename, argc, argv, "GW_BASENAME", -1, "-", ctl->
00639
      gw basename);
00640
         /* Output of atmospheric data... */
scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
00641
00642
00643
         scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00644
         ctl->atm dt out =
           scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00645
         ctl->atm filter
00646
00647
           (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
00648
00649
           (int) scan_ctl(filename, argc, argv, "ATM_BIN", -1, "0", NULL);
00650
         /* Output of CSI data... */
00651
        scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
00652
      csi_basename);
00653
        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",
00654
00655
00656
                   ctl->csi_obsfile);
00657
         ctl->csi obsmin =
00658
           scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00659
         ctl->csi modmin =
00660
           scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
         ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_ZO", -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);
00661
00662
00663
00664
         ctl->csi lon0 =
         scan_ctl(filename, argc, argv, "CSI_LONO", -1, "-180", NULL);
ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
00665
00666
00667
         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);
00668
                                                                                    '-90", NULL);
00669
00670
         ctl->csi_ny =
00671
00672
           (int) scan ctl(filename, argc, argv, "CSI NY", -1, "180", NULL);
00673
00674
         /\star Output of ensemble data... \star/
         scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
00675
      ens basename);
00676
00677
         /* Output of grid data... */
00678
         scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
00679
                   ctl->grid_basename);
        scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
00680
      grid_gpfile);
00681
        ctl->grid dt out =
00682
           scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
00683
         ctl->grid_sparse
           (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
00684
         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);
00685
00686
00687
         ctl->grid nz =
00688
           (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00689
         ctl->grid lon0 =
00690
           scan_ctl(filename, argc, argv, "GRID_LONO", -1, "-180", NULL);
00691
         ctl->grid lon1
00692
           scan ctl(filename, argc, argv, "GRID LON1", -1, "180", NULL);
00693
         ctl->grid nx =
00694
           (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00695
         ctl->grid lat0 =
00696
           scan_ctl(filename, argc, argv, "GRID_LATO", -1, "-90", NULL);
00697
         ctl->grid_lat1 =
           scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
00698
00699
         ctl->grid nv =
00700
           (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
00701
00702
         /* Output of profile data... */
         scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
00703
00704
                   ctl->prof_basename);
         scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
00705
      prof_obsfile);
00706
         ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
00707
         ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
00708
         ctl->prof_nz =
           (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
00709
00710
         ct.1->prof lon0 =
00711
           scan ctl(filename, argc, argv, "PROF LONO", -1, "-180", NULL);
         ctl->prof_lon1
00712
00713
           scan ctl(filename, argc, argv, "PROF LON1", -1, "180", NULL);
00714
         ctl->prof_nx =
00715
           (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00716
         ctl->prof lat0 =
00717
           scan ctl(filename, argc, argv, "PROF LATO", -1, "-90", NULL);
```

```
00718
         ctl->prof_lat1 =
00719
             scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
00720
          ctl->prof_ny =
00721
            (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00722
00723
          /* Output of station data... */
         scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
00725
                      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);
00726
00727
00728
00729 }
```

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.

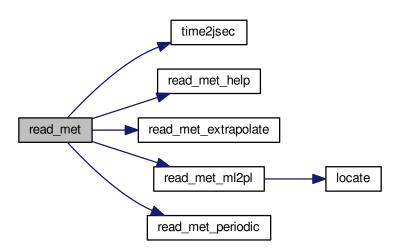
Definition at line 733 of file libtrac.c.

```
00736
00737
00738
       char cmd[LEN], levname[LEN], tstr[10];
00739
00740
       static float help[EX * EY];
00741
00742
       int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00743
00744
       size t np, nx, nv;
00745
00746
       /\star Write info... \star/
00747
       printf("Read meteorological data: %s\n", filename);
00748
00749
       /* Get time from filename... */
00750
       sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00751
       year = atoi(tstr);
00752
       sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00753
       mon = atoi(tstr);
00754
       sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00755
       dav = atoi(tstr);
00756
       sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00757
       hour = atoi(tstr);
00758
       time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00759
00760
        /* Open netCDF file... */
       if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
00761
00762
00763
          /* Try to stage meteo file... */
00764
         STOP_TIMER(TIMER_INPUT);
         00765
00766
00767
00768
00769
00770
             ERRMSG("Error while staging meteo data!");
00771
00772
         STOP_TIMER(TIMER_STAGE);
00773
         START_TIMER (TIMER_INPUT);
00774
00775
          /* Try to open again... */
00776
         NC(nc_open(filename, NC_NOWRITE, &ncid));
```

```
00777
00778
         /* Get dimensions... */
NC(nc_inq_dimid(ncid, "lon", &dimid));
NC(nc_inq_dimlen(ncid, dimid, &nx));
if (nx < 2 || nx > EX)
00779
00780
00781
00782
00783
            ERRMSG("Number of longitudes out of range!");
00784
00785
          NC(nc_inq_dimid(ncid, "lat", &dimid));
          NC(nc_inq_dimlen(ncid, dimid, &ny));
00786
00787
          if (ny < 2 || ny > EY)
00788
            ERRMSG("Number of latitudes out of range!");
00789
00790
          sprintf(levname, "lev");
00791
          NC(nc_inq_dimid(ncid, levname, &dimid));
00792
          NC(nc_inq_dimlen(ncid, dimid, &np));
00793
          if (np == 1) {
00794
            sprintf(levname, "lev_2");
            NC(nc_inq_dimid(ncid, levname, &dimid));
00796
            NC(nc_inq_dimlen(ncid, dimid, &np));
00797
          if (np < 2 || np > EP)
00798
            ERRMSG("Number of levels out of range!");
00799
00800
00801
          /* Store dimensions... */
         met->np = (int) np;
met->nx = (int) nx;
00803
00804
          met->ny = (int) ny;
00805
00806
          /* Get horizontal grid... */
          NC(nc_inq_varid(ncid, "lon", &varid));
00807
          NC(nc_get_var_double(ncid, varid, met->lon));
NC(nc_inq_varid(ncid, "lat", &varid));
80800
00809
00810
          NC(nc_get_var_double(ncid, varid, met->lat));
00811
          /* 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);
00812
00813
         read_met_help(ncid, "v", "V", 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->ho, 1.608f);
read_met_help(ncid, "o3", "O3", met, met->o3, 0.602f);
00815
00816
00817
00818
00819
00820
          /* Meteo data on pressure levels... */
00821
          if (ctl->met_np <= 0) {</pre>
00822
00823
             /* Read pressure levels from file... */
00824
            NC(nc_inq_varid(ncid, levname, &varid));
            NC(nc_get_var_double(ncid, varid, met->p));
for (ip = 0; ip < met->np; ip++)
  met->p[ip] /= 100.;
00825
00826
00828
00829
             /\star Extrapolate data for lower boundary... \star/
00830
            read_met_extrapolate(met);
00831
00832
00833
          /* Meteo data on model levels... */
00834
00835
            /* Read pressure data from file... */
read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00836
00837
00838
00839
             /* Interpolate from model levels to pressure levels... */
00840
            read_met_ml2pl(ctl, met, met->t);
00841
            read_met_ml2pl(ctl, met, met->u);
00842
            read_met_ml2pl(ctl, met, met->v);
            read_met_ml2pl(ctl, met, met->w);
read_met_ml2pl(ctl, met, met->h2o);
00843
00844
00845
            read_met_ml2pl(ctl, met, met->o3);
00846
00847
             /* Set pressure levels... */
            met->np = ctl->met_np;
for (ip = 0; ip < met->np; ip++)
00848
00849
00850
               met \rightarrow p[ip] = ctl \rightarrow met_p[ip];
00851
00852
00853
          /* Check ordering of pressure levels... */
          for (ip = 1; ip < met->np; ip++)
  if (met->p[ip - 1] < met->p[ip])
00854
00855
00856
               ERRMSG("Pressure levels must be descending!");
00857
          00858
00859
00860
            NC(nc_get_var_float(ncid, varid, help));
00861
            for (iy = 0; iy < met->ny; iy++)
for (ix = 0; ix < met->nx; ix++)
00862
00863
```

```
00866
         NC(nc_get_var_float(ncid, varid, help));
00867
00868
         for (iy = 0; iy < met->ny; iy++)
for (ix = 0; ix < met->nx; ix++)
00869
             met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00871
00872
         for (ix = 0; ix < met->nx; ix++)
           for (iy = 0; iy < met->ny; iy++)
  met->ps[ix][iy] = met->p[0];
00873
00874
00875
00876
       /* Create periodic boundary conditions... */
00877
       read_met_periodic(met);
00878
00879
        /* Close file... */
00880
       NC(nc_close(ncid));
00881 }
```

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 885 of file libtrac.c.

```
00886
                       {
00887
00888
        int ip, ip0, ix, iy;
00889
00890
         /* Loop over columns... */
00891
        for (ix = 0; ix < met->nx; ix++)
00892
           for (iy = 0; iy < met->ny; iy++) {
00893
             /* Find lowest valid data point... */
00894
             for (ip0 = met->np - 1; ip0 >= 0; ip0--)
   if (!gsl_finite(met->t[ix][iy][ip0])
00895
00896
00897
                   || !gsl_finite(met->u[ix][iy][ip0])
00898
                    || !gsl_finite(met->v[ix][iy][ip0])
00899
                   || !gsl_finite(met->w[ix][iy][ip0]))
00900
                 break;
00901
00902
             /* Extrapolate... */
```

```
for (ip = ip0; ip >= 0; ip--) {
00904
             met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
              met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
00905
              met \rightarrow v[ix][iy][ip] = met \rightarrow v[ix][iy][ip + 1];
00906
              met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
00907
              met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
00908
              met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00910
00911
          }
00912 }
```

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 916 of file libtrac.c.

```
00922
                     {
00923
00924
        static float help[EX * EY * EP];
00926
        int ip, ix, iy, n = 0, varid;
00927
00928
        /* Check if variable exists... */
        if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00929
00930
          if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00931
             return;
00932
        /* Read data... */
00933
00934
        NC(nc_get_var_float(ncid, varid, help));
00935
00936
        /* Copy and check data... */
        for (ip = 0; ip < met->np; ip++)
00937
00938
          for (iy = 0; iy < met->ny; iy++)
             for (ix = 0; ix < met->nx; ix++) {
  dest[ix][iy][ip] = scl * help[n++];
00939
00940
               if (fabs(dest[ix][iy][ip] / scl) > 1e14)
  dest[ix][iy][ip] = GSL_NAN;
00941
00942
00943
             }
00944 }
```

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 948 of file libtrac.c.

```
00951
00952
00953
        double aux[EP], p[EP], pt;
00954
00955
        int ip, ip2, ix, iy;
00956
00957
        /* Loop over columns... */
00958
        for (ix = 0; ix < met->nx; ix++)
00959
          for (iy = 0; iy < met->ny; iy++) {
00960
00961
            /* Copy pressure profile... */
for (ip = 0; ip < met->np; ip++)
00962
              p[ip] = met->pl[ix][iy][ip];
00963
00964
00965
            /* Interpolate... */
            for (ip = 0; ip < ctl->met_np; ip++) {
00966
00967
              pt = ctl->met_p[ip];
00968
              if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
               pt = p[0];
00969
00970
              else if ((pt > p[met->np - 1] && p[1] > p[0])
                || (pt < p[met->np - 1] && p[1] < p[0]))
pt = p[met->np - 1];
00971
00972
              00973
00974
00975
00976
00977
            /* Copy data... */
for (ip = 0; ip < ctl->met_np; ip++)
00978
00979
00980
              var[ix][iy][ip] = (float) aux[ip];
00981
00982 }
```

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 986 of file libtrac.c.

```
00987
00988
00989
       int ip, iy;
00990
00991
       /* Check longitudes... */
       if (!(fabs(met->lon[met->nx - 1] - met->lon[0])
00992
                  + met->lon[1] - met->lon[0] - 360) < 0.01))
00994
00995
00996
       /\star Increase longitude counter... \star/
       if ((++met->nx) > EX)
00997
00998
         ERRMSG("Cannot create periodic boundary conditions!");
00999
       01000
01001
     lon[0];
01002
01003
        /* Loop over latitudes and pressure levels... */
       for (iy = 0; iy < met->ny; iy++)
01004
        for (ip = 0; ip < met->np; ip++) {
01005
           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];
01006
01007
01008
           met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
01009
           01010
01011
01012
01013
01014 }
```

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 1018 of file libtrac.c.

```
01025
01026
       FILE *in = NULL:
01027
01028
       char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
01029
01030
        msg[LEN], rvarname[LEN], rval[LEN];
01031
01032
       int contain = 0, i;
01033
01034
       /* Open file... */
01035
       if (filename[strlen(filename) - 1] != '-')
         if (!(in = fopen(filename, "r")))
```

```
ERRMSG("Cannot open file!");
01038
01039
          /* Set full variable name... */
          if (arridx >= 0) {
01040
           sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
01041
01042
01043
         } else {
          sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
01044
01045
01046
01047
01048
         /* Read data... */
01049
          if (in != NULL)
          while (fgets(line, LEN, in))
if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
if (strcasecmp(rvarname, fullname1) == 0 ||
strcasecmp(rvarname, fullname2) == 0) {
01050
01051
01052
01053
01054
                    contain = 1;
01055
                    break;
01056
                 }
01057
         for (i = 1; i < argc - 1; i++)</pre>
01058
          if (strcasecmp(argv[i], fullname1) == 0 ||
               strcasecmp(argv[i], fullname2) == 0) {
sprintf(rval, "%s", argv[i + 1]);
01059
01060
01061
               contain = 1;
01062
               break;
01063
01064
01065
         /* Close file... */
         if (in != NULL)
01066
01067
            fclose(in);
01068
01069
         /* Check for missing variables... */
01070
          if (!contain) {
          if (strlen(defvalue) > 0)
   sprintf(rval, "%s", defvalue);
01071
01072
01073
            else {
01074
              sprintf(msg, "Missing variable %s!\n", fullname1);
01075
               ERRMSG(msg);
01076
01077
01078
01079
         /* Write info... */
         printf("%s = %s\n", fullname1, rval);
01080
01081
01082
          /* Return values... */
         if (value != NULL)
    sprintf(value, "%s", rval);
01083
01084
01085
          return atof(rval);
01086 }
```

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 1090 of file libtrac.c.

```
01098
01099
01100
       struct tm t0, t1;
01101
       t0.tm_year = 100;
01102
01103
        t0.tm_mon = 0;
01104
        t0.tm_mday = 1;
01105
        t0.tm\_hour = 0;
       t0.tm_min = 0;
t0.tm_sec = 0;
01106
01107
01108
01109
        t1.tm_year = year - 1900;
        t1.tm_mon = mon - 1;
01111
        t1.tm_mday = day;
01112
        t1.tm_hour = hour;
01113
        t1.tm_min = min;
01114
        t1.tm sec = sec:
01115
01116
       *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01117 }
```

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

Measure wall-clock time.

Definition at line 1121 of file libtrac.c.

```
01124
01125
01126
       static double starttime[NTIMER], runtime[NTIMER];
01127
01128
       /* Check id...
01129
        if (id < 0 || id >= NTIMER)
         ERRMSG("Too many timers!");
01130
01131
01132
        /* Start timer... */
01133
       if (mode == 1) {
        if (starttime[id] <= 0)</pre>
01134
01135
            starttime[id] = omp_get_wtime();
01136
            ERRMSG("Timer already started!");
01137
01138
01139
01140
       /* Stop timer... */
01141
        else if (mode == 2) {
01142
         if (starttime[id] > 0) {
01143
            runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
            starttime[id] = -1;
01144
01145
01146
       }
01147
01148
        /* Print timer... */
       else if (mode == 3)
printf("%s = %.3f s\n", name, runtime[id]);
01149
01150
01151 }
```

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

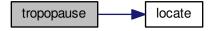
Definition at line 1155 of file libtrac.c.

```
01157
01158
01159
                     static double doys[12]
                    = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01160
01161
                     static double lats[73]
01162
                         = \{ -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -70, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5, -87.5
01163
                          -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,
01164
01165
                          -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,
01166
01167
01168
                           75, 77.5, 80, 82.5, 85, 87.5, 90
01169
01170
01171
01172
                     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,
01173
01174
                                          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,
01176
01177
                                         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,
01178
01179
                                         275.3, 275.6, 275.4, 274.1, 273.5},
01180
                     {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,
01181
01182
01183
                         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,
01184
                        98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193, 220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8, 284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01185
01186
01187
01188
                         287.5, 286.2, 285.8},
01189
                      {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1,
01190
                         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,
01191
                        100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2, 99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
01192
01193
01194
                         186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
```

```
279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
          304.3, 304.9, 306, 306.6, 306.2, 306}, {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
01196
01197
01198
           290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
01199
           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,
01200
           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,
01202
01203
           263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
          315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1}, {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8, 260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
01204
01205
01206
           205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7,
                                                                              104.1,
01207
                                                                                        102.7.
           101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
01208
01209
           102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
01210
           165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2
01211
           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},
01212
          {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
01214
           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,
01215
01216
           105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
           106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6, 127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245, 251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
01217
01218
01220
           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,
01221
01222
           187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
01223
           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,
01224
01225
           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
01227
01228
           275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
01229
          {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6,
          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,
01230
01231
           110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
01233
           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,
01234
01235
           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},
01236
          {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,
01237
01238
           243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
01239
01240
           114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4,
                                                                                 110.5
01241
           110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
           114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9, 203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
01242
01243
           276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
01244
          {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
01246
01247
           237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
           111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7, 106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
01248
01249
           112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7, 206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01250
           279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
01252
           305.1},
01253
          {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01254
01255
           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,
01256
           102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
01258
01259
           109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
01260
           241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
01261
           286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6}
          284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
01262
01263
           175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1, 100.4, 99.96, 99.6, 99.37, 99.32, 99.31, 99.46, 99.77, 100.2 100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
01265
01266
           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,
01267
01268
01269
           281.7, 281.1, 281.2}
01270
01271
01272
         double doy, p0, p1, pt;
01273
01274
         int imon, ilat:
01275
          /* Get day of year... */
         doy = fmod(t / 86400., 365.25);
01277
01278
         while (doy < 0)
01279
            doy += 365.25;
01280
01281
         /* Get indices... */
```

```
01282
      imon = locate(doys, 12, doy);
01283
      ilat = locate(lats, 73, lat);
01284
01285
      /* Get tropopause pressure... */
      01286
01287
01288
01289
01290
      pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01291
01292
      /* Return tropopause pressure... */
01293
      return pt;
01294 }
```

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 1298 of file libtrac.c.

```
01302
                    {
01303
        FILE *in, *out;
01304
01305
01306
        char line[LEN];
01307
01308
        double r, t0, t1;
01309
01310
        int ip, iq, year, mon, day, hour, min, sec;
01311
01312
        /\star Set time interval for output... \star/
01313
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01314
01315
01316
         /\star Check if gnuplot output is requested... \star/
01317
        if (ctl->atm_gpfile[0] != '-') {
01318
01319
           /* Write info... */
          printf("Plot atmospheric data: %s.png\n", filename);
01320
01321
          /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
01322
01323
01324
            ERRMSG("Cannot create pipe to gnuplot!");
01325
           01326
01327
01328
01329
           /* Set time string... */
           jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01330
01331
                   year, mon, day, hour, min);
01332
01333
01334
           /* Dump gnuplot file to pipe... */
01335
           if (!(in = fopen(ctl->atm_gpfile, "r")))
01336
             ERRMSG("Cannot open file!");
           while (fgets(line, LEN, in))
fprintf(out, "%s", line);
01337
01338
01339
           fclose(in);
01340
01341
```

```
01342
       else {
01343
01344
          /* Write info... */
         printf("Write atmospheric data: sn'', filename);
01345
01346
01347
          /* Create file... */
01348
         if (!(out = fopen(filename, "w")))
01349
           ERRMSG("Cannot create file!");
01350
01351
       /* Write binary data... */
if (ctl->atm_bin && ctl->atm_gpfile[0] == '-') {
01352
01353
         FWRITE(&atm->np, int,
01354
01355
01356
                out);
01357
         FWRITE(atm->time, double,
01358
                  (size_t) atm->np,
01359
                out);
01360
         FWRITE(atm->p, double,
01361
                  (size_t) atm->np,
01362
                out);
         FWRITE(atm->lon, double,
01363
01364
                  (size_t) atm->np,
01365
         out);
FWRITE(atm->lat, double,
01366
01367
                  (size_t) atm->np,
01368
                out);
01369
         for (iq = 0; iq < ctl->nq; iq++)
01370
           FWRITE(atm->q[iq], double,
01371
                    (size_t) atm->np,
01372
                  out);
01373
       }
01374
01375
       /* Write ASCII data... */
01376
       else {
01377
01378
          /* Write header... */
01379
         fprintf(out,
01380
                 "# $1 = time [s] \n"
                 "# \$2 = altitude [km]\n"
"# \$3 = longitude [deg]\n" "# \$4 = latitude [deg]\n");
01381
01382
         01383
01384
01385
         fprintf(out, "\n");
01386
01387
          /* Write data... */
01388
          for (ip = 0; ip < atm->np; ip++) {
01389
01390
01391
            /* Check time... */
01392
           if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
01393
01394
           01395
01396
01397
01398
01399
01400
             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01401
01402
           fprintf(out, "\n");
01403
01404
01405
01406
        /* Close file... */
01407
       fclose(out);
01408 }
```

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 1412 of file libtrac.c.

```
01416
                   {
01417
01418
        static FILE *in, *out;
01419
01420
        static char line[LEN]:
01421
        static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01422
          rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01424
01425
        static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01426
01427
        /* Init... */
01428
        if (!init) {
01429
          init = 1;
01430
01431
           /* Check quantity index for mass... */
01432
          if (ctl->qnt_m < 0)
            ERRMSG("Need quantity mass to analyze CSI!");
01433
01434
01435
           /* Open observation data file... */
01436
          printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
01437
           if (!(in = fopen(ctl->csi_obsfile, "r")))
            ERRMSG("Cannot open file!");
01438
01439
01440
          /* Create new file... */
          /* Create new file... */
printf("Write CSI data: %s\n", filename);
if (!/out = fopen(filename, "w")))
01441
01443
            ERRMSG("Cannot create file!");
01444
01445
           /* Write header... */
01446
          fprintf(out,
                   "# $1 = time [s]\n"
01447
                   "# $2 = number of hits (cx) \n"
01448
01449
                   "# $3 = number of misses (cy) \n"
01450
                   "# $4 = number of false alarms (cz)\n"
                   "# $5 = number of observations (cx + cy) \n"
01451
                   "# $6 = number of forecasts (cx + cz) n"
01452
                   "# $7 = bias (forecasts/observations) [%%]\n"
01453
                   "# $8 = probability of detection (POD) [%%]\n"
"# $9 = false alarm rate (FAR) [%%]\n"
01454
01455
01456
                   "# $10 = critical success index (CSI) [%%]\n\n");
01457
01458
01459
        /* Set time interval... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01460
01462
        /\star Initialize grid cells... \star/
01463
01464
        for (ix = 0; ix < ctl->csi_nx; ix++)
          for (iz = 0; iy < ctl->csi_ny; iy++)
for (iz = 0; iz < ctl->csi_nz; iz++)
01465
01466
               modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01467
01468
01469
        /* Read data... */
01470
        while (fgets(line, LEN, in)) {
01471
01472
           /* Read data... *
          if (sscanf(line, "%lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01473
01474
01475
             continue;
01476
          /* Check time... */
01477
01478
          if (rt < t0)</pre>
01479
            continue;
01480
           if (rt > t1)
01481
            break;
01482
           /* Calculate indices... */
01483
          ix = (int) ((rlon - ctl->csi_lon0))
01484
                        / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01485
01486
          iy = (int) ((rlat - ctl -> csi_lat0))
01487
                         / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
          iz = (int) ((rz - ctl -> csi_z0)
01488
01489
                        / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01490
01491
          /* Check indices... */
01492
           if (ix < 0 || ix >= ctl->csi_nx ||
```

```
iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01494
01495
01496
          /* Get mean observation index... */
01497
          obsmean[ix][iy][iz] += robs;
          obscount[ix][iy][iz]++;
01498
01499
01500
01501
        /* Analyze model data... ∗/
01502
        for (ip = 0; ip < atm->np; ip++) {
01503
01504
          /* Check time... */
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
01505
01506
01507
          /* Get indices... */
ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01508
01509
                       / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01510
          iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
01511
                       / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01512
          01513
01514
01515
          /* Check indices... */
if (ix < 0 || ix >= ctl->csi_nx ||
01516
01517
              iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01518
01519
            continue;
01520
01521
          /\star Get total mass in grid cell... \star/
         modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01522
01523
01524
01525
        /* Analyze all grid cells... */
01526
        for (ix = 0; ix < ctl->csi_nx; ix++)
         for (iy = 0; iy < ctl->csi_ny; iy++)
for (iz = 0; iz < ctl->csi_nz; iz++) {
01527
01528
01529
              /* Calculate mean observation index... */
01531
              if (obscount[ix][iy][iz] > 0)
01532
                obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01533
              /* Calculate column density... */
01534
              if (modmean[ix][iy][iz] > 0) {
  dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
01535
01536
                dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
01537
01538
                lat = ctl->csi_lat0 + dlat * (iy + 0.5);
                area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180. * cos(lat * M_PI / 180.);
01539
01540
                modmean[ix][iy][iz] /= (1e6 * area);
01541
01542
01544
              /* Calculate CSI... */
01545
              if (obscount[ix][iy][iz] > 0) {
               if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
    modmean[ix][iy][iz] >= ctl->csi_modmin)
01546
01547
01548
                  cx++;
                else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01550
                          modmean[ix][iy][iz] < ctl->csi_modmin)
01551
                01552
01553
01554
                  cz++;
              }
01556
01557
01558
       /* Write output... */
01559
        if (fmod(t, ctl->csi_dt_out) == 0) {
01560
          01561
01562
                  (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,

(cx + cz > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
01563
01564
01565
01566
                   (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01567
01568
01569
          /* Set counters to zero... */
         cx = cy = cz = 0;
01570
01571
01572
01573
       /* Close file... */
        if (t == ctl->t_stop)
01575
          fclose(out);
01576 }
```

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

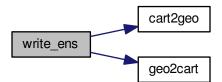
Write ensemble data.

Definition at line 1580 of file libtrac.c.

```
01584
                   {
01585
01586
        static FILE *out;
01587
        static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
01588
          t0, t1, x[NENS][3], xm[3];
01589
01590
        static int init, ip, iq;
01592
01593
        static size_t i, n;
01594
01595
        /* Init... */
01596
        if (!init) {
01597
          init = 1;
01598
01599
           /* Check quantities... */
          if (ctl->qnt_ens < 0)
   ERRMSG("Missing ensemble IDs!");</pre>
01600
01601
01602
01603
           /* Create new file... */
01604
          printf("Write ensemble data: %s\n", filename);
           if (!(out = fopen(filename, "w")))
01605
            ERRMSG("Cannot create file!");
01606
01607
01608
           /* Write header... */
01609
          fprintf(out,
                   "# $1 = time [s] \n"
                   "# \$2 = altitude [km]\n"
"# \$3 = longitude [deg]\n" "# \$4 = latitude [deg]\n");
01611
01612
          01613
01614
01615
          01616
01618
          fprintf(out, "# \$%d = number of members\n\n", 5 + 2 * ctl->nq);
01619
01620
01621
01622
        /* Set time interval... */
01623
        t0 = t - 0.5 * ctl->dt_mod;
        t1 = t + 0.5 * ctl -> dt_mod;
01624
01625
        /* Init... */
01626
01627
        ens = GSL_NAN;
        n = 0;
01628
        /* Loop over air parcels... */
for (ip = 0; ip < atm->np; ip++) {
01630
01631
01632
01633
          /* Check time... */
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
01634
01635
           continue;
01636
01637
           /\star Check ensemble id... \star/
          if (atm->q[ctl->qnt_ens][ip] != ens) {
01638
01639
01640
            /* Write results... */
01641
            if (n > 0) {
01642
01643
               /* Get mean position... */
               for (i = 0; i < n; i++) {
    xm[0] += x[i][0] / (double) n;
    xm[1] += x[i][1] / (double) n;</pre>
01644
01645
01646
01647
                 xm[2] += x[i][2] / (double) n;
01649
               cart2geo(xm, &dummy, &lon, &lat);
fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
01650
01651
01652
                       lat);
01653
01654
               /* Get quantity statistics... */
               for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01655
01656
01657
                 fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01658
               for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01659
01660
```

```
fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01662
                 fprintf(out, " %lu\n", n);
01663
              }
01664
01665
              /* Init new ensemble... */
01666
              ens = atm->q[ctl->qnt_ens][ip];
01667
01668
              n = 0;
01669
01670
01671
            /* Save data... */
01672
            p[n] = atm->p[ip];
01673
            geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
01674
            for (iq = 0; iq < ctl->nq; iq++)
            q[iq][n] = atm->q[iq][ip];
if ((++n) >= NENS)
    ERRMSG("Too many data points!");
01675
01676
01677
01678
01679
01680
         /* Write results... */
01681
          if (n > 0) {
01682
           /* Get mean position... */
xm[0] = xm[1] = xm[2] = 0;
for (i = 0; i < n; i++) {
  xm[0] += x[i][0] / (double) n;
  xm[1] += x[i][1] / (double) n;</pre>
01683
01684
01685
01686
01687
              xm[2] += x[i][2] / (double) n;
01688
01689
            cart2geo(xm, &dummy, &lon, &lat);
fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
01690
01691
01692
01693
            /* Get quantity statistics... */
            for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01694
01695
              fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01696
01697
            for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01698
01699
01700
               fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01701
            fprintf(out, " %lu\n", n);
01702
01703
01704
01705
          /* Close file... */
01706
          if (t == ctl->t_stop)
01707
            fclose(out);
01708 }
```

Here is the call graph for this function:



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

Write gridded data.

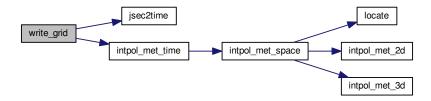
Definition at line 1712 of file libtrac.c.

```
01718
                     {
01719
01720
        FILE *in, *out;
01721
01722
         char line[LEN1:
01723
01724
        static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01725
           area, rho_air, press, temp, cd, mmr, t0, t1, r;
01726
01727
        static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01728
01729
         /* Check dimensions... */
         if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
    ERRMSG("Grid dimensions too large!");
01730
01731
01732
01733
         /\star Check quantity index for mass... \star/
01734
         if (ctl->ant m < 0)
           ERRMSG("Need quantity mass to write grid data!");
01735
01737
         /* Set time interval for output... */
        t0 = t - 0.5 * ctl->dt_mod;

t1 = t + 0.5 * ctl->dt_mod;
01738
01739
01740
         /* Set grid box size... */
dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
01741
01742
01743
01744
         dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
01745
01746
         /* Initialize grid... */
01747
         for (ix = 0; ix < ctl->grid_nx; ix++)
          for (iy = 0; iy < ctl->grid_ny; iy++)
    for (iz = 0; iz < ctl->grid_nz; iz++)
01748
01750
                grid_m[ix][iy][iz] = 0;
01751
         /* Average data... */
for (ip = 0; ip < atm->np; ip++)
   if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
01752
01753
01754
01755
01756
             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);
01757
01758
01759
01760
01761
              /* Check indices... */
01762
              if (ix < 0 || ix >= ctl->grid_nx ||
01763
                  iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01764
                continue;
01765
             /* Add mass... */
grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01766
01767
01768
01769
01770
        /* Check if gnuplot output is requested... */
01771
         if (ctl->grid_gpfile[0] != '-') {
01772
01773
           /* Write info... */
01774
           printf("Plot grid data: %s.png\n", filename);
01775
           /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
01776
01777
             ERRMSG("Cannot create pipe to gnuplot!");
01778
01779
01780
            /* Set plot filename...
01781
           fprintf(out, "set out \"%s.png\"\n", filename);
01782
01783
           /* Set time string... */
01784
           01785
01786
                    year, mon, day, hour, min);
01788
            /* Dump gnuplot file to pipe... */
           if (!(in = fopen(ctl->grid_gpfile, "r")))
01789
             ERRMSG("Cannot open file!");
01790
01791
           while (fgets(line, LEN, in))
fprintf(out, "%s", line);
01792
01793
           fclose(in);
01794
01795
01796
         else {
01797
01798
           /* Write info... */
           printf("Write grid data: %s\n", filename);
01800
            /* Create file... */
01801
01802
           if (!(out = fopen(filename, "w")))
             ERRMSG("Cannot create file!");
01803
01804
```

```
01805
01806
          /* Write header... */
         01807
01808
                    "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n"
01809
01810
                    "# $4 = latitude [deg]\n"
01811
01812
                    "# $5 = surface area [km^2]\n"
01813
                    "# $6 = layer width [km] \n"
                    "# \$7 = temperature [K]\n"
01814
                    "# $8 = column density [kg/m^2]\n"
01815
                    "# $9 = mass mixing ratio [1]\n\n");
01816
01817
          /* Write data... */
01818
01819
         for (ix = 0; ix < ctl->grid_nx; ix++) {
           if (ix > 0 && ctl->grid_nx, lx++) {
   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)
01820
01821
01822
01823
                 fprintf(out, "\n");
               for (iz = 0; iz < ctl->grid_nz; iz++)
01825
01826
                 if (!ctl->grid_sparse
                      | | ix == 0 | | iy == 0 | | iz == 0 | | grid_m[ix][iy][iz] > 0  {
01827
01828
01829
                   /* Set coordinates... */
                   z = ctl->grid_z0 + dz * (iz + 0.5);
01830
                   lon = ctl->grid_lon0 + dlon * (ix + 0.5);
lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01831
01832
01833
01834
                    /\!\star Get pressure and temperature... */\!
01835
                    press = P(z);
01836
                    intpol_met_time(met0, met1, t, press, lon, lat,
01837
                                       NULL, &temp, NULL, NULL, NULL, NULL, NULL);
01838
01839
                    /* Calculate surface area... */
                   area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
 * cos(lat * M_PI / 180.);
01840
01841
01842
01843
                    /* Calculate column density... */
01844
                   cd = grid_m[ix][iy][iz] / (1e6 * area);
01845
                   /* Calculate mass mixing ratio... */
rho_air = 100. * press / (287.058 * temp);
mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01846
01847
01848
01849
                    /* Write output... */
01850
01851
                    fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g,",
01852
                              t, z, lon, lat, area, dz, temp, cd, mmr);
01853
                 }
01854
            }
01855
         }
01856
01857
         /* Close file... */
01858
         fclose(out);
01859 }
```

Here is the call graph for this function:



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

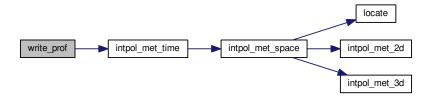
Write profile data.

Definition at line 1863 of file libtrac.c.

```
{
01870
01871
         static FILE *in, *out;
01872
01873
        static char line[LEN];
01874
        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,
01876
01877
          press, temp, rho_air, mmr, h2o, o3;
01878
01879
        static int init, obscount[GX][GY], ip, ix, iy, iz;
01880
01881
         /* Init... */
01882
        if (!init) {
01883
           init = 1;
01884
01885
           /\star Check quantity index for mass... \star/
           if (ctl->qnt_m < 0)
01886
             ERRMSG("Need quantity mass!");
01887
01888
01889
           /* Check dimensions...
           if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
01890
             ERRMSG("Grid dimensions too large!");
01891
01892
01893
           /* Open observation data file... */
           printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01894
01895
           if (!(in = fopen(ctl->prof_obsfile, "r")))
01896
             ERRMSG("Cannot open file!");
01897
01898
           /* Create new file... */
           printf("Write profile data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01899
01900
01901
             ERRMSG("Cannot create file!");
01902
01903
           /* Write header... */
           fprintf(out,
    "# $1
01904
                            = time [s]\n"
01905
                           = altitude [km]\n"
01906
01907
                    "# $3
                           = longitude [deg]\n"
01908
                    "# $4
                           = latitude [deg]\n"
                           = pressure [hPa]\n"
01909
                     "# $5
                     "# $6 = temperature [K] \n"
01910
                     "# $7 = mass mixing ratio [1]\n"
01911
                    "# $8 = H2O volume mixing ratio [1]\n"
01912
01913
                    "# $9 = 03 volume mixing ratio [1]\n"
01914
                     "# $10 = mean BT index [K] \n");
01915
01916
           /* 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;
01917
01918
01919
01920
01921
        /* Set time interval... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01922
01923
01924
01926
         /* Initialize... */
01927
         for (ix = 0; ix < ctl->prof_nx; ix++)
01928
           for (iy = 0; iy < ctl->prof_ny; iy++) {
             obsmean[ix][iy] = 0;
01929
             obscount[ix][iy] = 0;
01930
             tmean[ix][iy] = 0;
for (iz = 0; iz < ctl->prof_nz; iz++)
01931
01932
01933
               mass[ix][iy][iz] = 0;
01934
01935
        /* Read data... */
01936
        while (fgets(line, LEN, in)) {
01937
01939
            /* Read data... */
           if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01940
01941
             continue;
01942
01943
           /* Check time... */
01944
           if (rt < t0)
01945
             continue;
01946
           if (rt > t1)
01947
             break;
01948
           /* Calculate indices... */
01949
           ix = (int) ((rlon - ctl->prof_lon0) / dlon);
01950
01951
           iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01952
           /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01953
01954
01955
             continue:
```

```
01956
01957
              /* Get mean observation index... */
             obsmean[ix][iy] += robs;
tmean[ix][iy] += rt;
01958
01959
             obscount[ix][iy]++;
01960
01961
01962
01963
           /★ Analyze model data... ★/
01964
           for (ip = 0; ip < atm->np; ip++) {
01965
01966
              /\star Check time... \star/
01967
             if (atm->time[ip] < t0 || atm->time[ip] > t1)
01968
               continue;
01969
01970
             /* Get indices... */
             ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01971
01972
01973
01974
             /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx ||
    iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01975
01976
01977
01978
                continue;
01979
01980
              /* Get total mass in grid cell... */
01981
             mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01982
01983
01984
          /* Extract profiles... */
          for (ix = 0; ix < ctl->prof_nx; ix++)
  for (iy = 0; iy < ctl->prof_ny; iy++)
01985
01986
01987
                if (obscount[ix][iy] > 0) {
01988
01989
                   /* Write output... */
01990
                   fprintf(out, "\n");
01991
                   /* Loop over altitudes... */
01992
01993
                   for (iz = 0; iz < ctl->prof_nz; iz++) {
01994
01995
                     /* 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);
01996
01997
01998
01999
02000
                      /* Get meteorological data... */
02001
                     press = P(z);
                     intpol_met_time(met0, met1, t, press, lon, lat,
    NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
02002
02003
02004
                     /* 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.);
02005
02006
02007
02008
02009
                     mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
02010
02011
                      /* Write output... */
02012
                     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
02013
                                tmean[ix][iy] / obscount[ix][iy],
                                z, lon, lat, press, temp, mmr, h2o, o3, obsmean[ix][iy] / obscount[ix][iy]);
02014
02015
02016
02017
02018
02019
          /* Close file... */
02020
          if (t == ctl->t_stop)
02021
             fclose(out);
02022 1
```

Here is the call graph for this function:



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

Write station data.

Definition at line 2026 of file libtrac.c.

```
02030
                   {
02031
02032
       static FILE *out;
02034
       static double rmax2, t0, t1, x0[3], x1[3];
02035
02036
       static int init, ip, iq;
02037
02038
        /* Init... */
02039
       if (!init) {
02040
          init = 1;
02041
02042
         /* Write info... */
         printf("Write station data: sn", filename);
02043
02044
02045
          /* Create new file... */
02046
          if (!(out = fopen(filename, "w")))
02047
            ERRMSG("Cannot create file!");
02048
02049
          /* Write header... */
02050
          fprintf(out,
                  "# $1 = time [s]\n"
"# $2 = altitude [km]\n"
02051
02052
02053
                  "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
          02054
02055
02056
02057
          fprintf(out, "\n");
02058
02059
          /* Set geolocation and search radius... */
02060
          geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
02061
          rmax2 = gsl_pow_2(ctl->stat_r);
02062
02063
02064
        /* Set time interval for output... */
02065
        t0 = t - 0.5 * ctl->dt_mod;
        t1 = t + 0.5 * ctl->dt_mod;
02066
02067
        /* Loop over air parcels... */
for (ip = 0; ip < atm->np; ip++) {
02068
02069
02070
02071
          /* Check time... */
02072
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
02073
            continue;
02074
02075
          /* Check station flag... */
if (ctl->qnt_stat >= 0)
02076
02077
           if (atm->q[ctl->qnt_stat][ip])
02078
02079
02080
          /* Get Cartesian coordinates... */
02081
          geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
02082
02083
          /\star Check horizontal distance... \star/
```

```
02084
          if (DIST2(x0, x1) > rmax2)
02085
            continue;
02086
          /* Set station flag... */
if (ctl->qnt_stat >= 0)
  atm->q[ctl->qnt_stat][ip] = 1;
02087
02088
02089
02090
02091
          /* Write data... */
          02092
02093
          for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02094
02095
02096
            fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02097
02098
          fprintf(out, "\n");
02099
02100
        /* Close file... */
if (t == ctl->t_stop)
02101
02102
02103
          fclose(out);
02104 }
```

Here is the call graph for this function:



```
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) 2013-2018 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00028
00029 void cart2geo(
00030
       double *x,
       double *z,
00031
00032
       double *lon,
00033
       double *lat) {
00034
00035
       double radius:
00036
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
00043 /
       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
00087
      return dv * 180. / (M PI * RE);
00088 }
00089
00091
00092 double dz2dp(
00093
     double dz,
     double p) {
00094
00095
00096
     return -dz * p / H0;
00097 }
00098
00100
00101 void geo2cart(
00102
00103
      double lon,
     double lat,
00104
00105
     double *x) {
00106
00107
     double radius;
00108
00109
      radius = z + RE;
     x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
x[2] = radius * sin(lat / 180 * M_PI);
00110
00111
00112
00113 }
00114
00116
00117 void get_met(
     ctl_t * ctl,
char *metbase,
00118
00119
00120
     double t,
00121
      met_t * met0,
00122
     met_t * met1) {
00123
00124
     char filename[LEN1:
00125
00126
     static int init;
00127
00128
      /* Init... */
00129
      if (!init) {
00130
       init = 1;
00131
```

```
00132
         get_met_help(t, -1, metbase, ctl->dt_met, filename);
00133
         read_met(ctl, filename, met0);
00134
00135
         get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
     dt_met, filename);
    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,
00186
       double wx,
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
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
```

```
/* 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)
00277
          intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
00278
        if (h2o != NULL)
00279
          intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
00280
        if (o3 != NULL)
00281
         intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00282 }
00283
00285
00286 void intpol_met_time(
       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,
                       w == NULL ? NULL : &w0,
00309
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
00316
                       w == NULL ? NULL : &w1,
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;
       if (t != NULL)
00325
00326
         *t = wt * (t0 - t1) + t1;
       if (u != NULL)
00327
00328
         *11 = wt * (110 - 111) + 111:
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(
00377
       double *xx,
00378
       int n,
00379
       double x) {
00380
00381
       int i, ilo, ihi;
00382
00383
       ilo = 0;
       ihi = n - 1;
00384
       i = (ihi + ilo) >> 1;
00385
00386
00387
       if (xx[i] < xx[i + 1])
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
               ihi = i;
00399
              else
00400
                ilo = i;
00401
           }
00402
00403
        return ilo;
00404 }
00405
00407
00408 void read_atm(
        const char *filename,
ctl_t * ctl,
00409
00410
00411
        atm_t * atm) {
00412
00413
        FILE *in;
00414
        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 binary data... */
00430
         if (ctl->atm_bin && ctl->atm_gpfile[0] == '-') {
00431
          FREAD(&atm->np, int,
00432
                  1.
00433
                  in);
           FREAD (atm->time, double,
00434
00435
                    (size_t) atm->np,
                  in);
00436
00437
           FREAD(atm->p, double,
00438
                    (size_t) atm->np,
00439
                  in);
           FREAD(atm->lon, double,
00440
                    (size_t) atm->np,
00441
00442
                  in);
00443
           FREAD(atm->lat, double,
00444
                    (size_t) atm->np,
           00445
00446
00447
00448
00449
00450
00451
        /* Read ASCII data... */
00452
00453
        else {
00454
00455
            /* Read line... */
00456
           while (fgets(line, LEN, in)) {
00457
             /* 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]);
00458
00459
00460
00461
00462
00463
00464
00465
00466
              /\star Convert altitude to pressure... \star/
00467
              atm \rightarrow p[atm \rightarrow np] = P(atm \rightarrow p[atm \rightarrow np]);
00468
              /* Increment data point counter... */ if ((++atm->np) > NP)
00469
00470
                ERRMSG("Too many data points!");
00471
00472
00473
         }
00474
00475
         /* Close file... */
00476
         fclose(in);
00477
00478
         /* Check number of points... */
```

```
if (atm->np < 1)
        ERRMSG("Can not read any data!");
00480
00481 }
00482
00484
00485 void read_ctl(
00486
       const char *filename,
00487
        int argc,
       char *argv[],
ctl_t * ctl) {
00488
00489
00490
00491
        int ip, iq;
00492
00493
        /* Write info... */
        \label{linear_printf} $$ printf("\nAssive-Parallel Trajectory Calculations (MPTRAC)\n" (executable: %s | compiled: %s, %s)\n'n", 
00494
00495
                argv[0], __DATE__, __TIME__);
00496
00497
00498
        /* Initialize quantity indices... */
00499
        ctl->qnt_ens = -1;
        ctl \rightarrow qnt_m = -1;
00500
        ctl->qnt_r = -1;
00501
00502
        ctl->qnt_rho = -1;
00503
        ctl->qnt_ps = -1;
        ctl->qnt_p = -1;
00504
00505
        ctl->qnt_t = -1;
00506
        ctl->qnt_u = -1;
00507
        ctl->qnt_v = -1;
        ctl->qnt_w = -1;
00508
00509
        ctl->qnt_h2o = -1;
00510
        ctl \rightarrow qnt_o3 = -1;
00511
        ctl->qnt_theta = -1;
00512
        ctl->qnt\_pv = -1;
        ctl->qnt_tice = -1;
ctl->qnt_tsts = -1;
00513
00514
00515
        ctl->qnt tnat = -1;
        ctl->qnt_gw_var = -1;
00517
        ctl \rightarrow qnt_stat = -1;
00518
00519
        /* Read quantities... */
        \label{eq:ctl-nq}  \mbox{ctl->nq = (int) } \mbox{scan\_ctl(filename, argc, argv, "NQ", -1, "0", NULL);} 
00520
        if (ctl->ng > NQ)
00521
00522
          ERRMSG("Too many quantities!");
        for (iq = 0; iq < ctl->nq; iq++) {
00523
00524
          00525
00526
00527
00528
          /* Try to identify quantity... */
if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
00530
00531
00532
           ctl->qnt_ens = iq;
            sprintf(ctl->qnt_unit[iq], "-");
00533
          } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
ctl->qnt_m = iq;
00534
00536
            sprintf(ctl->qnt_unit[iq], "kg");
00537
          } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
            ctl->qnt_r = iq;
00538
            sprintf(ctl->qnt_unit[iq], "m");
00539
          } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
00540
            ctl->qnt_rho = iq;
00541
00542
            sprintf(ctl->qnt_unit[iq], "kg/m^3");
00543
          } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
00544
            ctl->qnt_ps = iq;
            sprintf(ctl->qnt_unit[iq], "hPa");
00545
          } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
ctl->qnt_p = iq;
00546
00547
            sprintf(ctl->qnt_unit[iq], "hPa");
00549
          } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
00550
            ctl->qnt_t = iq;
            sprintf(ctl->qnt_unit[iq], "K");
00551
          } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
ctl->qnt_u = iq;
00552
00553
            sprintf(ctl->qnt_unit[iq], "m/s");
00554
00555
          } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
00556
          ctl->qnt_v = iq;
            sprintf(ctl->qnt_unit[iq], "m/s");
00557
          sprint(ctr >qnc_unit(iq), m/s',')
else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
ctl->qnt_w = iq;
sprintf(ctl->qnt_unit[iq], "hPa/s");
00558
00559
00560
00561
          } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
00562
            ct1->qnt_h2o = iq;
00563
           sprintf(ctl->qnt_unit[iq], "1");
          } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
ctl->qnt_o3 = iq;
00564
00565
```

```
sprintf(ctl->qnt_unit[iq], "1");
           } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
00567
00568
             ctl->qnt_theta = iq;
             sprintf(ctl->qnt_unit[iq], "K");
00569
           } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
  ctl->qnt_pv = iq;
  sprintf(ctl->qnt_unit[iq], "PVU");
00570
00571
00572
           } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
00573
            ctl->qnt_tice = iq;
00574
00575
             sprintf(ctl->qnt_unit[iq], "K");
00576
           } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
00577
            ctl->qnt_tsts = iq;
             sprintf(ctl->qnt_unit[iq], "K");
00578
00579
           } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
00580
             ctl->qnt_tnat = iq;
             sprintf(ctl->qnt_unit[iq], "K");
00581
           } else if (strcmp(ctl->qnt_name[iq], "gw_var") == 0) {
  ctl->qnt_gw_var = iq;
  sprintf(ctl->qnt_unit[iq], "K^2");
00582
00583
00585
           } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
00586
              ctl->qnt_stat = iq;
00587
             sprintf(ctl->qnt_unit[iq], "-");
00588
           } else
             scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00589
00590
00591
00592
         /* Time steps of simulation... */
00593
         ctl->direction =
         (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
if (ctl->direction != -1 && ctl->direction != 1)
00594
00595
          ERRMSG("Set DIRECTION to -1 or 1!");
00596
00597
        ctl->t_start
00598
           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);
00599
00600
00601
00602
         /* Meteorological data... */
         ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
00603
00604
         ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
00605
         if (ctl->met_np > EP)
00606
          ERRMSG("Too many levels!");
         for (ip = 0; ip < ctl->met_np; ip++)
  ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
00607
00608
00609
00610
00611
         /* Isosurface parameters... */
00612
         ctl->isosurf
        = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00613
00614
00615
00616
         /* Diffusion parameters... */
00617
        ctl->turb_dx_trop
00618
           = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00619
         ctl->turb dx strat
           = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00620
00621
         ctl->turb dz trop
00622
            = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00623
         ctl->turb dz strat
00624
           = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00625
         ctl->turb meso :
           scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00626
00627
00628
        /* Life time of particles... */
        ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL); ctl->tdec_strat =
00629
00630
00631
           scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00632
00633
         /* PSC analysis... */
        ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
00634
        ctl->psc_hno3 =
00635
00636
          scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
00637
        /* Gravity wave analysis... */
scan_ctl(filename, argc, argv, "GW_BASENAME", -1, "-", ctl->
00638
00639
      qw basename);
00640
00641
         /* Output of atmospheric data... */
        scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
00642
      atm_basename);
00643
         scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00644
         ctl->atm dt out
00645
           scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
         ctl->atm_filter
00646
00647
           (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
00648
         ctl->atm_bin =
           (int) scan_ctl(filename, argc, argv, "ATM_BIN", -1, "0", NULL);
00649
00650
```

```
/* Output of CSI data... */
         scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
       csi_basename);
00653 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",
00654
00655
                   ctl->csi_obsfile);
00657
         ctl->csi_obsmin =
00658
           scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00659
         ctl->csi modmin =
           scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
00660
         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);
00661
00662
00663
00664
         ctl->csi_lon0 =
         scan_ctl(filename, argc, argv, "CSI_LONO", -1, "-180", NULL);
ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
00665
00666
00667
         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);
00668
00669
00670
00671
         ctl->csi_ny =
           (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
00672
00673
00674
         /* Output of ensemble data... */
         scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
00675
      ens_basename);
00676
00677
         /* Output of grid data... */
         00678
00679
00680
         scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
      grid_gpfile);
00681
        ctl->grid_dt_out =
00682
           scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
00683
         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);
00684
00686
00687
         ctl->grid_nz =
00688
           (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00689
         ctl->grid lon0 =
           scan_ctl(filename, argc, argv, "GRID_LONO", -1, "-180", NULL);
00690
00691
         ctl->grid_lon1 =
00692
           scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
00693
         ctl->grid_nx =
00694
           (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00695
         ctl->grid lat0 =
           scan ctl(filename, argc, argv, "GRID LATO", -1, "-90", NULL);
00696
         ctl->grid_lat1 =
00697
00698
           scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
00699
         ctl->grid_ny =
00700
            (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
00701
00702
         /* Output of profile data... */
00703
         scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
                    ctl->prof_basename);
00704
         scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
00705
      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);
00706
00707
00708
         ctl->prof_nz =
            (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
00710
         ctl->prof lon0 =
00711
           scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
00712
         ctl->prof_lon1 =
           scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
00713
00714
         ctl->prof nx =
00715
           (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00716
         ctl->prof_lat0 =
00717
            scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
         ctl->prof_lat1 =
00718
00719
           scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
00720
         ctl->prof_ny =
00721
           (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00722
00723
         /* Output of station data... */
         scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
00724
00725
                    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);
00726
00727
00728
00729 }
00730
00732
00733 void read met (
```

```
00734
         ctl_t * ctl,
00735
         char *filename,
00736
         met_t * met) {
00737
         char cmd[LEN], levname[LEN], tstr[10];
00738
00739
00740
         static float help[EX * EY];
00741
00742
         int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00743
00744
         size_t np, nx, ny;
00745
00746
         /* Write info... */
00747
         printf("Read meteorological data: %s\n", filename);
00748
         /* Get time from filename... */
sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00749
00750
00751
         year = atoi(tstr);
         sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00752
00753
         mon = atoi(tstr);
00754
         sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00755
         day = atoi(tstr);
00756
         sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00757
         hour = atoi(tstr);
00758
         time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00759
         /\star Open netCDF file... \star/
00760
00761
         if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
00762
00763
            /* Try to stage meteo file... */
00764
           STOP_TIMER(TIMER_INPUT);
00765
            START_TIMER(TIMER_STAGE);
            if (ctl->met_stage[0] != '-') {
    sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
00766
00767
              year, mon, day, hour, filename);
if (system(cmd) != 0)
00768
00769
00770
                ERRMSG("Error while staging meteo data!");
00771
00772
            STOP_TIMER(TIMER_STAGE);
00773
            START_TIMER(TIMER_INPUT);
00774
00775
            /* Try to open again... */
00776
           NC(nc_open(filename, NC_NOWRITE, &ncid));
00777
00778
          /* Get dimensions... */
00779
00780
         NC(nc_inq_dimid(ncid, "lon", &dimid));
00781
         NC(nc_inq_dimlen(ncid, dimid, &nx));
00782
         if (nx < 2 || nx > EX)
    ERRMSG("Number of longitudes out of range!");
00783
00784
00785
         NC(nc_inq_dimid(ncid, "lat", &dimid));
00786
         NC(nc_inq_dimlen(ncid, dimid, &ny));
00787
         if (ny < 2 || ny > EY)
            ERRMSG("Number of latitudes out of range!");
00788
00789
00790
         sprintf(levname, "lev");
00791
         NC(nc_inq_dimid(ncid, levname, &dimid));
00792
         NC(nc_inq_dimlen(ncid, dimid, &np));
00793
         if (np == 1) {
            sprintf(levname, "lev_2");
00794
00795
           NC(nc_inq_dimid(ncid, levname, &dimid));
00796
           NC(nc_inq_dimlen(ncid, dimid, &np));
00797
00798
         if (np < 2 \mid \mid np > EP)
00799
           ERRMSG("Number of levels out of range!");
00800
00801
         /* Store dimensions... */
         met->np = (int) np;
00802
         met->nx = (int) nx;
00803
00804
         met->ny = (int) ny;
00805
         /* Get horizontal grid... */
NC(nc_inq_varid(ncid, "lon", &varid));
00806
00807
         NC(nc_get_var_double(ncid, varid, met->lon));
NC(nc_inq_varid(ncid, "lat", &varid));
80800
00809
00810
         NC(nc_get_var_double(ncid, varid, met->lat));
00811
         /* 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);
00812
00813
00814
         read_met_help(ncid, "v", "v", 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->ho, 1.608f);
read_met_help(ncid, "o3", "O3", met, met->o3, 0.602f);
00815
00816
00817
00818
00819
00820
         /* Meteo data on pressure levels... */
```

```
00821
        if (ctl->met_np <= 0) {</pre>
00822
00823
           /* Read pressure levels from file... */
00824
           NC(nc_inq_varid(ncid, levname, &varid));
           NC(nc_get_var_double(ncid, varid, met->p));
for (ip = 0; ip < met->np; ip++)
  met->p[ip] /= 100.;
00825
00826
00828
00829
           /* Extrapolate data for lower boundary... */
00830
           read_met_extrapolate(met);
00831
00832
00833
         /* Meteo data on model levels... */
00834
        else {
00835
           /* Read pressure data from file... */
read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00836
00837
00838
00839
           /* Interpolate from model levels to pressure levels... */
00840
           read_met_ml2pl(ctl, met, met->t);
00841
           read_met_ml2pl(ctl, met, met->u);
00842
           read_met_ml2pl(ctl, met, met->v);
00843
           read_met_ml2pl(ctl, met, met->w);
00844
           read_met_ml2pl(ctl, met, met->h2o);
00845
           read_met_ml2pl(ctl, met, met->o3);
00846
00847
           /* Set pressure levels... */
           met->np = ctl->met_np;
for (ip = 0; ip < met->np; ip++)
   met->p[ip] = ctl->met_p[ip];
00848
00849
00850
00851
00852
00853
         /* Check ordering of pressure levels... */
        for (ip = 1; ip < met->np; ip++)
  if (met->p[ip - 1] < met->p[ip])
    ERRMSG("Pressure levels must be descending!");
00854
00855
00856
00857
         /* Read surface pressure... */
        00859
00860
00861
           NC(nc_get_var_float(ncid, varid, help));
        00862
00863
00864
00865
00866
00867
           NC(nc_get_var_float(ncid, varid, help));
           for (iy = 0; iy < met->ny; iy++)
  for (ix = 0; ix < met->nx; ix++)
00868
00869
               met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00870
00871
        } else
00872
           for (ix = 0; ix < met->nx; ix++)
00873
             for (iy = 0; iy < met->ny; iy++)
00874
               met->ps[ix][iy] = met->p[0];
00875
00876
        /* Create periodic boundary conditions... */
00877
        read_met_periodic(met);
00878
00879
         /* Close file... */
00880
        NC(nc_close(ncid));
00881 }
00882
00883 /***
00884
00885 void read_met_extrapolate(
00886 met_t * met) {
00887
00888
        int ip, ip0, ix, iv;
00889
00890
         /* Loop over columns... */
00891
         for (ix = 0; ix < met->nx; ix++)
00892
           for (iy = 0; iy < met->ny; iy++) {
00893
00894
             /* Find lowest valid data point... */
             for (ip0 = met->np - 1; ip0 >= 0; ip0--)
   if (!gsl_finite(met->t[ix][iy][ip0])
00895
00896
00897
                    || !gsl_finite(met->u[ix][iy][ip0])
00898
                    || !gsl_finite(met->v[ix][iy][ip0])
00899
                    || !gsl_finite(met->w[ix][iy][ip0]))
00900
                  break:
00901
00902
             /* Extrapolate... */
00903
             for (ip = ip0; ip >= 0; ip--) {
00904
               met \rightarrow t[ix][iy][ip] = met \rightarrow t[ix][iy][ip + 1];
               met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
00905
00906
               met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
00907
```

```
met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
00909
             met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00910
00911
         }
00912 }
00913
00915
00916 void read_met_help(
00917
       int ncid,
       char *varname,
char *varname2,
00918
00919
       met_t * met,
00920
00921
       float dest[EX][EY][EP],
00922
       float scl) {
00923
       static float help[EX * EY * EP];
00924
00925
00926
       int ip, ix, iy, n = 0, varid;
00927
       /* Check if variable exists... */
00928
00929
       if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
        if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00930
00931
           return:
00932
00933
       /* Read data... */
00934
       NC(nc_get_var_float(ncid, varid, help));
00935
00936
       /* 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++) {
00937
00938
00939
00940
             dest[ix][iy][ip] = scl * help[n++];
00941
             if (fabs(dest[ix][iy][ip] / scl) > 1e14)
00942
               dest[ix][iy][ip] = GSL_NAN;
00943
00944 }
00947
00948 void read_met_ml2pl(
00949
       ctl_t * ctl,
met t * met,
00950
00951
       float var[EX][EY][EP]) {
00952
00953
       double aux[EP], p[EP], pt;
00954
00955
       int ip, ip2, ix, iy;
00956
00957
       /* Loop over columns... */
00958
       for (ix = 0; ix < met->nx; ix++)
00959
         for (iy = 0; iy < met->ny; iy++) {
00960
           /* Copy pressure profile... */
for (ip = 0; ip < met->np; ip++)
  p[ip] = met->pl[ix][iy][ip];
00961
00962
00963
00964
00965
           /* Interpolate... */
00966
           for (ip = 0; ip < ctl->met_np; ip++) {
             pt = ctl->met_p[ip];
00967
             if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
00968
00969
              pt = p[0];
             else if ((pt > p[met->np - 1] && p[1] > p[0])
00970
               | | (pt < p[met->np - 1] && p[1] < p[0]))
pt = p[met->np - 1];
00971
00972
             ip2 = locate(p, met->np, pt);
00973
             00974
00975
00976
00977
00978
           /* Copy data... */
for (ip = 0; ip < ctl->met_np; ip++)
00979
00980
             var[ix][iy][ip] = (float) aux[ip];
00981
00982 }
00983
00985
00986 void read_met_periodic(
00987
       met_t * met) {
00988
00989
       int ip, iy;
00990
00991
       /* Check longitudes... */
00992
       + \text{ met} -> \text{lon}[1] - \text{met} -> \text{lon}[0] - 360) < 0.01))
00993
00994
         return:
```

```
00995
00996
          /* Increase longitude counter... */
00997
         if ((++met->nx) > EX)
           ERRMSG("Cannot create periodic boundary conditions!");
00998
00999
01000
         /* Set longitude... */
         met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
01001
01002
01003
          /* 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];
01004
01005
01006
              met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
01007
01008
              met->v[met->nx - 1][iy][ip] = met->w[0][iy][ip];
met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01009
01010
01011
01012
01013
01014 }
01015
01017
01018 double scan_ctl(
         const char *filename,
01019
01020
          int argc,
01021
         char *argv[],
01022
         const char *varname,
01023
         int arridx.
01024
         const char *defvalue.
01025
         char *value) {
01026
01027
         FILE *in = NULL;
01028
         char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
01029
01030
          msg[LEN], rvarname[LEN], rval[LEN];
01031
01032
         int contain = 0, i;
01033
01034
          /* Open file... */
         if (filename[strlen(filename) - 1] != '-')
if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");
01035
01036
01037
01038
01039
         /* Set full variable name... */
01040
         if (arridx >= 0) {
           sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
01041
01042
01043
         } else {
           sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
01044
01045
01046
01047
01048
         /* Read data... */
01049
         if (in != NULL)
           while (fgets(line, LEN, in))
01050
01051
              if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
                if (strcasecmp(rvarname, fullname1) == 0 ||
01052
01053
                      strcasecmp(rvarname, fullname2) == 0) {
01054
                   contain = 1;
01055
                   break;
01056
                }
01057
         for (i = 1; i < argc - 1; i++)
01058
           if (strcasecmp(argv[i], fullname1) == 0 ||
              strcasecmp(argv[i], fullname2) == 0) {
sprintf(rval, "%s", argv[i + 1]);
01059
01060
01061
              contain = 1;
01062
              break:
01063
           }
01064
01065
         /* Close file... */
01066
         if (in != NULL)
           fclose(in);
01067
01068
         /\star Check for missing variables... \star/
01069
01070
         if (!contain) {
          if (strlen(defvalue) > 0)
   sprintf(rval, "%s", defvalue);
01071
01072
01073
            else (
01074
              sprintf(msg, "Missing variable %s!\n", fullname1);
01075
              ERRMSG(msg);
01076
01077
01078
        /* Write info... */
printf("%s = %s\n", fullname1, rval);
01079
01080
```

```
01081
01082
       /* Return values... */
01083
       if (value != NULL)
        sprintf(value, "%s", rval);
01084
01085
       return atof(rval);
01086 }
01087
01089
01090 void time2jsec(
01091
       int year,
01092
       int mon.
01093
        int day,
01094
       int hour,
01095
        int min,
01096
       int sec,
01097
       double remain,
01098
       double *jsec) {
01099
01100
       struct tm t0, t1;
01101
01102
       t0.tm_year = 100;
01103
       t0.tm\_mon = 0;
       t0.tm mday = 1;
01104
01105
       t0.tm_hour = 0;
       t0.tm_min = 0;
01106
01107
       t0.tm\_sec = 0;
01108
01109
       t1.tm_year = year - 1900;
01110
       t1.tm_mon = mon - 1;
01111
       t1.tm mdav = dav;
01112
       t1.tm_hour = hour;
01113
       t1.tm_min = min;
01114
       t1.tm_sec = sec;
01115
       *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01116
01117 }
01118
01120
01121 void timer(
01122
      const char *name,
01123
       int id.
01124
       int mode) {
01125
01126
       static double starttime[NTIMER], runtime[NTIMER];
01127
       /* Check id... */
if (id < 0 || id >= NTIMER)
01128
01129
01130
         ERRMSG("Too many timers!");
01131
01132
        /* Start timer... */
01133
       if (mode == 1) {
01134
        if (starttime[id] <= 0)</pre>
01135
           starttime[id] = omp_get_wtime();
01136
         else
01137
           ERRMSG("Timer already started!");
01138
01139
       /* Stop timer... */
else if (mode == 2) {
  if (starttime[id] > 0) {
    runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
01140
01141
01142
01143
01144
            starttime[id] = -1;
01145
01146
       }
01147
       /* Print timer... */
01148
       else if (mode == 3)
01149
         printf("%s = %.3f s\n", name, runtime[id]);
01150
01151 }
01152
01154
01155 double tropopause(
01156 double t,
01157
       double lat) {
01158
01159
       static double doys[12]
       = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01160
01161
01162
       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,
01163
01164
         -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,
01165
01166
01167
```

```
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
01169
01170
01171
01172
                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,
01173
                              175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
01175
                              99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54, 98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128, 152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275, 277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
01176
01177
01178
01179
                275.3, 275.6, 275.4, 274.1, 273.5}, 280.1, 275.3, 276.3, 276.3, 275.3, 275.3, 275.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 276.3, 2
01180
01181
01182
                  300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
                 150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42, 98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27, 98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193, 220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
01183
01184
01185
                  284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
                  287.5, 286.2, 285.8},
01188
01189
                {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,
01190
01191
                  100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01192
                  99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
01193
                  186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
01194
01195
                  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}, {306.2, 306.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4, 290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2, 195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
01196
01197
01198
01199
                  102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
01200
01201
                  99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
01202
                  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,
01203
                  315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
01204
                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,
01206
                  205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
01207
                 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, 273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01208
01209
01210
01212
                  325.3, 325.8, 325.8},
01213
                {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
01214
                  222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
01215
                  228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
                  105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8, 106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
01216
01217
                  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,
01219
01220
                  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, 110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
01221
01222
01223
                  111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
01225
                117. 9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9, 224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5, 275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8, {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4, 185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
01226
01227
01228
01229
                  233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
01231
01232
                  110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9
01233
                  112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
01234
                  120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
                  230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7, 278.2, 282.6, 287.4, 290.9, 292.5, 293},
01235
01236
                {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,
01238
01239
                  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, 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,
01240
01241
01242
01243
                  276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
01244
                {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,
01245
01246
                  237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2, 111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01247
01248
                  106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4,
                  112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
01250
01251
                  206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01252
                  279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
                  305.1},
01253
01254
                {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
```

```
253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
           223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9, 108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
01256
01257
01258
           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},
01259
01260
01261
          {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
01262
           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,
01263
01264
01265
01266
01267
01268
           280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
01269
           281.7, 281.1, 281.2}
01270
01271
01272
          double doy, p0, p1, pt;
01274
          int imon, ilat;
01275
01276
          /* Get day of year... */
          doy = fmod(t / 86400., 365.25);
while (doy < 0)
  doy += 365.25;</pre>
01277
01278
01279
01280
01281
          /* Get indices... */
         imon = locate(doys, 12, doy);
ilat = locate(lats, 73, lat);
01282
01283
01284
01285
          /* Get tropopause pressure... */
         p0 = LIN(lats[ilat], tps[imon][ilat],
lats[ilat + 1], tps[imon][ilat + 1], lat);
01286
01287
         01288
01289
         pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01290
01291
01292
         /* Return tropopause pressure... */
01293
          return pt;
01294 }
01295
01297
01298 void write_atm(
01299
         const char *filename,
01300
          ctl_t * ctl,
01301
         atm t * atm.
01302
         double t) {
01303
01304
         FILE *in, *out;
01305
01306
         char line[LEN];
01307
01308
         double r, t0, t1;
01309
01310
         int ip, iq, year, mon, day, hour, min, sec;
01311
01312
          /\star Set time interval for output... \star/
          t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01313
01314
01315
          /* Check if gnuplot output is requested... */
if (ctl->atm_gpfile[0] != '-') {
01316
01317
01318
01319
             /* Write info... */
01320
            printf("Plot atmospheric data: %s.png\n", filename);
01321
01322
            /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
01323
01324
               ERRMSG("Cannot create pipe to gnuplot!");
01325
01326
             /* Set plot filename... */
             fprintf(out, "set out \"%s.png\"\n", filename);
01327
01328
01329
             /* Set time string... */
             jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01330
01331
01332
                       year, mon, day, hour, min);
01333
             /\star Dump gnuplot file to pipe... \star/
01334
            if (!(in = fopen(ctl->atm_gpfile, "r")))
    ERRMSG("Cannot open file!");
01335
01336
             while (fgets(line, LEN, in))
fprintf(out, "%s", line);
01337
01338
01339
            fclose(in);
01340
01341
```

```
01342
       else {
01343
01344
         /* Write info... */
         printf("Write atmospheric data: sn'', filename);
01345
01346
01347
         /* Create file... */
        if (!(out = fopen(filename, "w")))
01348
01349
           ERRMSG("Cannot create file!");
01350
01351
01352
       /* Write binary data... */
       if (ctl->atm_bin && ctl->atm_gpfile[0] == '-') {
01353
01354
         FWRITE(&atm->np, int,
01355
01356
                out);
01357
         FWRITE(atm->time, double,
01358
                 (size_t) atm->np,
01359
                out);
01360
         FWRITE(atm->p, double,
01361
                 (size_t) atm->np,
01362
                out);
         FWRITE(atm->lon, double,
01363
01364
                 (size_t) atm->np,
01365
                out);
01366
         FWRITE(atm->lat, double,
01367
                 (size_t) atm->np,
01368
                out);
01369
        for (iq = 0; iq < ctl->nq; iq++)
01370
          FWRITE(atm->q[iq], double,
01371
                   (size_t) atm->np,
01372
                 out);
01373
       }
01374
01375
       /* Write ASCII data... */
01376
       else {
01377
01378
         /* Write header... */
01379
         fprintf(out,
01380
                 "# $1 = time [s] \n"
                "# \$2 = altitude [km]\n"
"# \$3 = longitude [deg]\n" "# \$4 = latitude [deg]\n");
01381
01382
         01383
01384
01385
01386
         fprintf(out, "\n");
01387
01388
         /* Write data... */
         for (ip = 0; ip < atm->np; ip++) {
01389
01390
01391
           /* Check time... */
01392
           if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
01393
01394
           01395
01396
01397
           for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01398
01399
01400
             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01401
           fprintf(out, "\n");
01402
01403
01404
01405
01406
       /* Close file... */
01407
       fclose(out);
01408 }
01409
01411
01412 void write_csi(
01413
       const char *filename,
       ctl_t * ctl,
atm_t * atm,
01414
01415
01416
       double t) {
01417
01418
       static FILE *in, *out;
01419
01420
       static char line[LEN]:
01421
       static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01422
01423
        rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01424
01425
       static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01426
       /* Init... */
if (!init) {
01427
01428
```

```
01429
          init = 1;
01430
01431
          /* Check quantity index for mass... */
          if (ctl->qnt_m < 0)
01432
            ERRMSG("Need quantity mass to analyze CSI!");
01433
01434
01435
          /* Open observation data file... */
01436
          printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
          if (!(in = fopen(ctl->csi_obsfile, "r")))
01437
01438
            ERRMSG("Cannot open file!");
01439
          /* Create new file... */
01440
          printf("Write CSI data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01441
01442
01443
            ERRMSG("Cannot create file!");
01444
          /* Write header... */
01445
          01446
01447
                   "# $2 = number of hits (cx) \n"
01448
01449
                   "# $3 = number of misses (cy) \n"
01450
                   "# $4 = number of false alarms (cz)\n"
                  "# $5 = number of observations (cx + cy)\n" # $6 = number of forecasts (cx + cz)\n"
01451
01452
                   "# $7 = bias (forecasts/observations) [%%]\n"
01453
                   "# $8 = probability of detection (POD) [%%]\n"
01454
01455
                   "# $9 = false alarm rate (FAR) [%%]\n"
01456
                  "# $10 = critical success index (CSI) [%%]\n\n");
01457
01458
01459
        /* Set time interval... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01460
01461
01462
01463
        /* Initialize grid cells... */
        for (ix = 0; ix < ctl->csi_nx; ix++)
  for (iy = 0; iy < ctl->csi_ny; iy++)
    for (iz = 0; iz < ctl->csi_nz; iz++)
01464
01465
01466
01467
              modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01468
01469
        /* Read data... */
01470
        while (fgets(line, LEN, in)) {
01471
01472
          /* Read data... */
01473
          if (sscanf(line, "%lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01474
01475
            continue:
01476
          /* Check time... */
01477
01478
          <u>if</u> (rt < t0)
01479
            continue;
01480
          if (rt > t1)
01481
            break;
01482
          /* Calculate indices... */
01483
          ix = (int) ((rlon - ctl->csi_lon0)
01484
                       / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01486
          iy = (int) ((rlat - ctl->csi_lat0))
01487
                       / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01488
          iz = (int) ((rz - ctl -> csi_z0)
                       / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01489
01490
01491
          /* Check indices... */
01492
          if (ix < 0 || ix >= ctl->csi_nx ||
01493
              iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01494
            continue;
01495
01496
          /* Get mean observation index... */
01497
          obsmean[ix][iy][iz] += robs;
01498
          obscount[ix][iy][iz]++;
01499
01500
01501
        /\star Analyze model data... \star/
        for (ip = 0; ip < atm->np; ip++) {
01502
01503
01504
          /* Check time... */
01505
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
01506
            continue;
01507
01508
          /* Get indices... */
          ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01509
                       / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01510
01511
          iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
01512
                       / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
          01513
01514
01515
```

```
/* Check indices... */
01517
          if (ix < 0 || ix >= ctl->csi_nx ||
               iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01518
01519
             continue;
01520
          /* Get total mass in grid cell... */
modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01521
01522
01523
01524
01525
         /* Analyze all grid cells... */
01526
        for (ix = 0; ix < ctl->csi_nx; ix++)
         for (iy = 0; iy < ctl->csi_ny; iy++)
    for (iz = 0; iz < ctl->csi_nz; iz++) {
01527
01528
01529
01530
                /* Calculate mean observation index... */
               if (obscount[ix][iy][iz] > 0)
  obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01531
01532
01533
01534
               /* Calculate column density... */
01535
               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;
01536
01537
                 lat = ctl->csi_lat0 + dlat * (iy + 0.5);
area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
* cos(lat * M_PI / 180.);
01538
01539
01540
                 modmean[ix][iy][iz] /= (1e6 * area);
01541
01542
01543
01544
               /* Calculate CSI... */
01545
               if (obscount[ix][iy][iz] > 0) {
                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01546
01547
                      modmean[ix][iy][iz] >= ctl->csi_modmin)
01548
01549
                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01550
                            modmean[ix][iy][iz] < ctl->csi_modmin)
                    cy++;
01551
01552
                 else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
                           modmean[ix][iy][iz] >= ctl->csi_modmin)
01553
01554
                    cz++;
01555
01556
01557
01558
        /* Write output... */
01559
        if (fmod(t, ctl->csi_dt_out) == 0) {
01560
01561
01562
          fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
                    (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN, (cx + cz > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
01563
01564
01565
01566
01567
                    (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01568
01569
           /* Set counters to zero... */
01570
          cx = cy = cz = 0;
01571
        }
01572
01573
        /* Close file... */
01574
        if (t == ctl->t_stop)
01575
          fclose(out);
01576 }
01577
01579
01580 void write_ens(
01581 const char *filename,
01582
        ctl_t * ctl,
atm_t * atm,
01583
01584
        double t) {
01586
        static FILE *out;
01587
        static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
t0, t1, x[NENS][3], xm[3];
01588
01589
01590
01591
        static int init, ip, iq;
01592
01593
        static size_t i, n;
01594
01595
        /* Tnit... */
01596
        if (!init) {
          init = 1;
01597
01598
01599
           /* Check quantities... */
01600
           if (ctl->qnt_ens < 0)</pre>
             ERRMSG("Missing ensemble IDs!");
01601
01602
```

```
/* Create new file... */
            printf("Write ensemble data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01604
01605
              ERRMSG("Cannot create file!");
01606
01607
01608
            /* Write header... */
01609
            fprintf(out,
01610
                      "# $1 = time [s] \n"
                      "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01611
01612
            for (iq = 0; iq < ctl->nq; iq++)
  fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
01613
01614
                         ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01615
            for (iq = 0; iq < ctl->nq; iq++)
fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
01616
01617
            01618
01619
01620
01621
01622
          /* Set time interval... */
         t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01623
01624
01625
01626
         /* Init... */
01627
         ens = GSL_NAN;
         n = 0;
01628
01629
         /* Loop over air parcels... */
for (ip = 0; ip < atm->np; ip++) {
01630
01631
01632
01633
            /* Check time... */
01634
            if (atm->time[ip] < t0 || atm->time[ip] > t1)
01635
01636
01637
             /* Check ensemble id... */
            if (atm->q[ctl->qnt_ens][ip] != ens) {
01638
01639
01640
              /* Write results... */
01641
              if (n > 0) {
01642
01643
                 /* Get mean position... */
                 xm[0] = xm[1] = xm[2] = 0;
for (i = 0; i < n; i++) {
  xm[0] += x[i][0] / (double) n;
  xm[1] += x[i][1] / (double) n;
  xm[2] += x[i][2] / (double) n;</pre>
01644
01645
01646
01647
01648
01649
                 cart2geo(xm, &dummy, &lon, &lat);
fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
01650
01651
01652
                           lat);
01653
01654
                  /* Get quantity statistics... */
                 for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01655
01656
                    fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01657
01658
                 for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01659
01660
01661
                    fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01662
                 fprintf(out, " lu\n", n);
01663
01664
01665
               /\star Init new ensemble... \star/
01666
01667
              ens = atm->q[ctl->qnt_ens][ip];
01668
              n = 0;
01669
01670
01671
            /* Save data... */
            p[n] = atm->p[ip];
01673
            geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
01674
            for (iq = 0; iq < ctl->nq; iq++)
            q[iq][n] = atm->q[iq][ip];
if ((++n) >= NENS)
01675
01676
              ERRMSG("Too many data points!");
01677
01678
01679
01680
          /* Write results... */
01681
         if (n > 0) {
01682
            /* Get mean position... */
01683
            for (i = 0; i < n; i++) {
    xm[0] += x[i][0] / (double) n;
    xm[1] += x[i][1] / (double) n;
    xm[2] += x[i][2] / (double) n;</pre>
01684
01685
01686
01687
01688
01689
```

```
cart2geo(xm, &dummy, &lon, &lat);
fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
01691
01692
01693
           /* Get quantity statistics... */
01694
           for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01695
             fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01696
01697
           for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01698
01699
            fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01700
01701
01702
           fprintf(out, " %lu\n", n);
01703
01704
01705
         /* Close file... */
         if (t == ctl->t_stop)
01706
01707
          fclose(out);
01708 }
01709
01711
01712 void write_grid(
01713
        const char *filename,
        ctl_t * ctl,
met_t * met0,
01714
01715
01716
        met_t * met1,
01717
        atm_t * atm,
01718
        double t) {
01719
01720
        FILE *in, *out;
01721
01722
        char line[LEN];
01723
01724
        static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01725
           area, rho_air, press, temp, cd, mmr, t0, t1, r;
01726
01727
        static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01728
01729
        /* Check dimensions... */
01730
        if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
    ERRMSG("Grid dimensions too large!");
01731
01732
         /\star Check quantity index for mass... \star/
01733
01734
        if (ctl->qnt_m < 0)</pre>
01735
          ERRMSG("Need quantity mass to write grid data!");
01736
        /* Set time interval for output... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01737
01738
01739
01740
01741
         /* Set grid box size...
01742
         dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
        dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
01743
01744
01745
01746
         /* Initialize grid... */
01747
        for (ix = 0; ix < ctl->grid_nx; ix++)
01748
         for (iy = 0; iy < ctl->grid_ny; iy++)
             for (iz = 0; iz < ctl->grid_nz; iz++)
01749
01750
               grid_m[ix][iy][iz] = 0;
01751
01752
         /* Average data... */
01753
         for (ip = 0; ip < atm->np; ip++)
01754
           if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {</pre>
01755
01756
              /* Get index... */
             ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
01757
01758
             iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01759
01760
01761
              /* Check indices... */
             if (ix < 0 || ix >= ctl->grid_nx ||
    iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01762
01763
01764
                continue;
01765
01766
              /* Add mass... */
01767
             grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01768
01769
01770
        /* Check if gnuplot output is requested... */
if (ctl->grid_gpfile[0] != '-') {
01771
01772
01773
           /* Write info... */
01774
           printf("Plot grid data: %s.png\n", filename);
01775
01776
           /* Create gnuplot pipe... */
```

```
if (!(out = popen("gnuplot", "w")))
01778
            ERRMSG("Cannot create pipe to gnuplot!");
01779
          01780
01781
01782
01783
          /* Set time string... */
          jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01784
01785
01786
                   year, mon, day, hour, min);
01787
01788
          /* Dump gnuplot file to pipe... */
01789
          if (!(in = fopen(ctl->grid_gpfile, "r")))
            ERRMSG("Cannot open file!");
01790
01791
          while (fgets(line, LEN, in))
            fprintf(out, "%s", line);
01792
01793
          fclose(in):
01794
        }
01795
01796
        else {
01797
01798
          /\star Write info... \star/
          printf("Write grid data: %s\n", filename);
01799
01800
01801
           /* Create file... */
          if (!(out = fopen(filename, "w")))
01802
01803
            ERRMSG("Cannot create file!");
01804
01805
        /* Write header... */
01806
01807
        fprintf(out,
01808
                 "# $1 = time [s]\n"
01809
                 "# $2 = altitude [km] \n"
01810
                 "# $3 = longitude [deg] \n"
                 "# $4 = latitude [deg] \n"
01811
                 "# $5 = surface area [km^2]\n"
01812
                 "# $6 = layer width [km] \n"
01813
                 "# $7 = temperature [K]\n"
01814
                "# $8 = column density [kg/m^2]\n"
"# $9 = mass mixing ratio [1]\n\n";
01815
01816
01817
        /* Write data... */
01818
        /* white data...,
for (ix = 0; ix < ctl->grid_nx; ix++) {
    if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01819
01820
            fprintf(out, "\n");
01821
01822
           for (iy = 0; iy < ctl->grid_ny; iy++) {
           if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
  fprintf(out, "\n");
for (iz = 0; iz < ctl->grid_nz; iz++)
01823
01824
01825
              if (!ctl->grid_sparse
01826
                   || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01827
01828
01829
                /* Set coordinates... */
                z = ctl->grid_z0 + dz * (iz + 0.5);
lon = ctl->grid_lon0 + dlon * (ix + 0.5);
01830
01831
                lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01832
01834
                 /* Get pressure and temperature... */
01835
                 press = P(z);
                 01836
01837
01838
01839
                 /* Calculate surface area... */
                area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
 * cos(lat * M_PI / 180.);
01840
01841
01842
01843
                /* Calculate column density... */
cd = grid_m[ix][iy][iz] / (1e6 * area);
01844
01845
                 /* Calculate mass mixing ratio... */
                rho_air = 100. * press / (287.058 * temp);
mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01847
01848
01849
                01850
01851
01852
01853
               }
01854
01855
01856
        /* Close file... */
01857
01858
       fclose(out);
01859 }
01860
01862
01863 void write prof(
```

```
01864
        const char *filename,
         ctl_t * ctl,
met_t * met0,
01865
01866
01867
         met_t * met1,
01868
         atm t * atm.
         double t) {
01869
01870
01871
         static FILE *in, *out;
01872
01873
        static char line[LEN];
01874
        static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
01875
          rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, mmr, h2o, o3;
01876
01877
01878
01879
         static int init, obscount[GX][GY], ip, ix, iy, iz;
01880
01881
         /* Init... */
         if (!init) {
01882
01883
           init = 1;
01884
01885
           /\star Check quantity index for mass... \star/
           if (ctl->qnt_m < 0)
01886
             ERRMSG("Need quantity mass!");
01887
01888
01889
           /* Check dimensions... */
01890
           if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
             ERRMSG("Grid dimensions too large!");
01891
01892
01893
           /* Open observation data file... */
01894
           printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01895
              (!(in = fopen(ctl->prof_obsfile, "r")))
01896
              ERRMSG("Cannot open file!");
01897
           /* Create new file... */
printf("Write profile data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01898
01899
01900
              ERRMSG("Cannot create file!");
01901
01902
01903
            /* Write header... */
01904
           fprintf(out,
                     "# $1 = time [s]\n"
"# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n"
01905
01906
01907
01908
                     "# $4
                            = latitude [deg]\n"
01909
                     "# $5
                            = pressure [hPa]\n"
01910
                     "# $6 = temperature [K] \n"
                     "# $7 = mass mixing ratio [1]\n"
01911
                     "# $8 = H2O volume mixing ratio [1]\n"
01912
01913
                     "# $9
                            = 03 volume mixing ratio [1]\n'
                     "# $10 = mean BT index [K]\n");
01914
01915
01916
           /* 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;
01917
01918
01919
01920
01921
01922
         /\star Set time interval... \star/
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01923
01924
01925
01926
         /* Initialize... */
01927
         for (ix = 0; ix < ctl->prof_nx; ix++)
01928
           for (iy = 0; iy < ctl->prof_ny; iy++) {
              obsmean[ix][iy] = 0;
01929
              obscount[ix][iy] = 0;
01930
              tmean[ix][iy] = 0;
for (iz = 0; iz < ctl->prof_nz; iz++)
01931
01932
                mass[ix][iy][iz] = 0;
01933
01934
01935
         /* Read data... */
while (fgets(line, LEN, in)) {
01936
01937
01938
01939
01940
           if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01941
             continue;
01942
           /* Check time... */
01943
01944
           if (rt < t0)</pre>
01945
             continue;
01946
           if (rt > t1)
01947
              break;
01948
01949
           /* Calculate indices... */
           ix = (int) ((rlon - ctl->prof_lon0) / dlon);
01950
```

```
iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01952
01953
           /* Check indices... */
01954
           if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01955
             continue;
01956
01957
           /\star Get mean observation index... \star/
01958
           obsmean[ix][iy] += robs;
          tmean[ix][iy] += rt;
01959
01960
          obscount[ix][iy]++;
01961
01962
01963
         /* Analyze model data... */
01964
        for (ip = 0; ip < atm->np; ip++) {
01965
           /* Check time... */
01966
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
01967
01968
            continue;
01969
01970
           /* Get indices... */
          ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01971
01972
01973
01974
01975
           /* Check indices... */
01976
          if (ix < 0 || ix >= ctl->prof_nx ||
01977
               iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
             continue;
01978
01979
          /* Get total mass in grid cell... */
mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01980
01981
01982
01983
01984
        /* Extract profiles... */
        for (ix = 0; ix < ctl->prof_nx; ix++)
  for (iy = 0; iy < ctl->prof_ny; iy++)
01985
01986
             if (obscount[ix][iy] > 0) {
01987
01988
01989
               /* Write output...
01990
               fprintf(out, "\n");
01991
01992
               /* Loop over altitudes... */
01993
               for (iz = 0; iz < ctl->prof nz; iz++) {
01994
01995
                 /* Set coordinates... */
01996
                 z = ctl->prof_z0 + dz * (iz + 0.5);
                 lon = ctl->prof_lon0 + dlon * (ix + 0.5);
lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01997
01998
01999
02000
                 /* Get meteorological data... */
02001
                 press = P(z);
02002
                 intpol_met_time(met0, met1, t, press, lon, lat,
02003
                                   NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
02004
                 /* 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.);
02005
02006
02007
02008
02009
                 mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
02010
                 02011
02012
02013
02014
                           z, lon, lat, press, temp, mmr, h2o, o3,
02015
                           obsmean[ix][iy] / obscount[ix][iy]);
02016
02017
             }
02018
02019
        /* Close file... */
        if (t == ctl->t_stop)
02021
          fclose(out);
02022 }
02023
02025
02026 void write_station(
        const char *filename,
02027
        ctl_t * ctl,
atm_t * atm,
02028
02029
02030
        double t) {
02031
02032
        static FILE *out;
02033
02034
        static double rmax2, t0, t1, x0[3], x1[3];
02035
02036
        static int init, ip, iq;
02037
```

```
02038
        /* Init... */
02039
        if (!init) {
02040
          init = 1;
02041
          /* Write info... */
02042
         printf("Write station data: %s\n", filename);
02043
02044
02045
          /\star Create new file... \star/
02046
          if (!(out = fopen(filename, "w")))
            ERRMSG("Cannot create file!");
02047
02048
02049
          /* Write header... */
02050
          fprintf(out,
                   "# $1 = time [s]\n"
02051
                  "# \$2 = altitude [km]\n"
"# \$3 = longitude [deg]\n" "# \$4 = latitude [deg]\n");
02052
02053
          02054
02055
02056
02057
          fprintf(out, "\n");
02058
02059
          /\star Set geolocation and search radius... \star/
02060
          geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
          rmax2 = gsl_pow_2(ctl->stat_r);
02061
02062
02063
02064
        /\star Set time interval for output... \star/
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
02065
02066
02067
02068
        /* Loop over air parcels... */
for (ip = 0; ip < atm->np; ip++) {
02069
02070
02071
          /\star Check time... \star/
02072
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
02073
            continue;
02074
02075
          /* Check station flag... */
02076
          if (ctl->qnt_stat >= 0)
02077
           if (atm->q[ctl->qnt_stat][ip])
02078
              continue;
02079
02080
          /* Get Cartesian coordinates... */
02081
          geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
02082
02083
          /\star Check horizontal distance... \star/
02084
          if (DIST2(x0, x1) > rmax2)
02085
            continue;
02086
02087
          /* Set station flag... */
          if (ctl->qnt_stat >= 0)
02088
02089
            atm->q[ctl->qnt_stat][ip] = 1;
02090
          02091
02092
02093
          for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02094
02095
02096
            fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02097
02098
          fprintf(out, "\n");
02099
02100
02101
        /* Close file... */
        if (t == ctl->t_stop)
02102
02103
          fclose(out);
02104 }
```

#### 5.15 libtrac.h File Reference

MPTRAC library declarations.

## **Data Structures**

· struct ctl t

Control parameters.

struct atm\_t

Atmospheric data.

struct met\_t

Meteorological data.

#### **Functions**

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

Convert Cartesian coordinates to geolocation.

double deg2dx (double dlon, double lat)

Convert degrees to horizontal distance.

double deg2dy (double dlat)

Convert degrees to horizontal distance.

double dp2dz (double dp, double p)

Convert pressure to vertical distance.

• double dx2deg (double dx, double lat)

Convert horizontal distance to degrees.

double dy2deg (double dy)

Convert horizontal distance to degrees.

• double dz2dp (double dz, double p)

Convert vertical distance to pressure.

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

Convert geolocation to Cartesian coordinates.

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

Get meteorological data for given timestep.

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

Get meteorological data for timestep.

• void intpol met 2d (double array[EX][EY], int ix, int iy, double wx, double wy, double \*var)

Linear interpolation of 2-D meteorological data.

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

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

Spatial interpolation of meteorological data.

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

Temporal interpolation of meteorological data.

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

Convert seconds to date.

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

Find array index.

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

Read atmospheric data.

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

Read control parameters.

void read met (ctl t \*ctl, char \*filename, met t \*met)

Read meteorological data file.

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

Extrapolate meteorological data at lower boundary.

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

Read and convert variable from meteorological data file.

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

Convert meteorological data from model levels to pressure levels.

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

Create meteorological data with periodic boundary conditions.

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

Read a control parameter from file or command line.

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

Convert date to seconds.

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

Measure wall-clock time.

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

Write atmospheric data.

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

Write CSI data.

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

Write ensemble data.

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

Write gridded data.

 $\bullet \ \ \text{void write\_prof (const char *filename, ctl\_t *ctl, met\_t *met0, met\_t *met1, atm\_t *atm, double t)}\\$ 

Write profile data.

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

Write station data.

### 5.15.1 Detailed Description

MPTRAC library declarations.

Definition in file libtrac.h.

## 5.15.2 Function Documentation

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

Convert Cartesian coordinates to geolocation.

Definition at line 29 of file libtrac.c.

```
00033 {
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;
```

### 5.15.2.2 double deg2dx ( double dlon, double lat )

Convert degrees to horizontal distance.

Definition at line 45 of file libtrac.c.

### 5.15.2.3 double deg2dy ( double dlat )

Convert degrees to horizontal distance.

Definition at line 54 of file libtrac.c.

# 5.15.2.4 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 }
```

### 5.15.2.5 double dx2deg ( double dx, double lat )

Convert horizontal distance to degrees.

Definition at line 71 of file libtrac.c.

## 5.15.2.6 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 }
```

```
5.15.2.7 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 }
```

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

Convert geolocation to Cartesian coordinates.

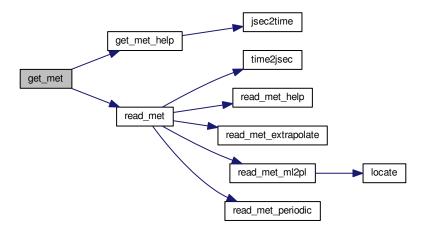
Definition at line 101 of file libtrac.c.

5.15.2.9 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 }
```



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

Get meteorological data for timestep.

Definition at line 156 of file libtrac.c.

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

Here is the call graph for this function:



5.15.2.11 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 }
```

5.15.2.12 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 }
```

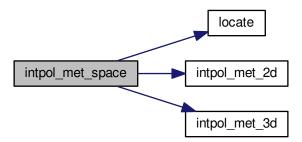
5.15.2.13 void intpol\_met\_space (  $met_t * met$ , double p, double lon, double lo

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 }
```



5.15.2.14 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 )

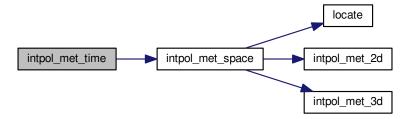
Temporal interpolation of meteorological data.

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:



5.15.2.15 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
        jsec0 = (time_t) jsec + timegm(&t0);
00362
00363
       t1 = gmtime(&jsec0);
00364
00365
        *year = t1->tm_year + 1900;
00366
        *mon = t1->tm_mon + 1;
00367
        *day = t1->tm_mday;
00368
        *hour = t1->tm_hour;
        *min = t1->tm_min;
00369
00370
        *sec = t1->tm_sec;
00371
        *remain = jsec - floor(jsec);
00372 }
```

```
5.15.2.16 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;
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
             if (xx[i] > x)
00390
00391
              ihi = i;
             else
00392
               ilo = i;
00393
00394
        } else
00395
         while (ihi > ilo + 1) {
00396
             i = (ihi + ilo) >> 1;
00397
             if (xx[i] \le x)
              ihi = i;
00398
00399
            else
00400
              ilo = i;
00401
00402
00403
        return ilo;
00404 }
```

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

Read atmospheric data.

Definition at line 408 of file libtrac.c.

```
00411
00412
       FILE *in;
00413
00414
00415
       char line[LEN], *tok;
00416
00417
00418
       /* Init... */
atm->np = 0;
00419
00420
00421
00422
       /* Write info... */
00423
       printf("Read atmospheric data: %s\n", filename);
00424
00425
       /* Open file... */
       if (!(in = fopen(filename, "r")))
00426
         ERRMSG("Cannot open file!");
00427
00428
00429
       /* Read binary data... */
       if (ctl->atm_bin && ctl->atm_gpfile[0] == '-') {
00430
00431
         FREAD(&atm->np, int,
00432
                1.
00433
                in);
          FREAD(atm->time, double,
00434
00435
                  (size_t) atm->np,
00436
                in);
00437
          FREAD(atm->p, double,
00438
                 (size_t) atm->np,
00439
                in);
          FREAD(atm->lon, double,
00440
00441
                 (size_t) atm->np,
00442
                in);
         FREAD(atm->lat, double,
00443
00444
                 (size_t) atm->np,
                in);
00445
00446
          for (iq = 0; iq < ctl->nq; iq++)
00447
           FREAD(atm->q[iq], double,
```

```
00448
                            (size_t) atm->np,
00449
                         in);
00450
00451
00452
          /* Read ASCII data... */
00453
          else {
00454
00455
              /* Read line... */
00456
              while (fgets(line, LEN, in)) {
00457
                /* 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]);
00458
00459
00460
00461
00462
00463
00464
00465
00466
                 /* Convert altitude to pressure... */
                atm \rightarrow p[atm \rightarrow np] = P(atm \rightarrow p[atm \rightarrow np]);
00467
00468
00469
                 /* Increment data point counter... */
                 if ((++atm->np) > NP)
00470
                   ERRMSG("Too many data points!");
00471
00472
             }
00473
00474
00475
           /* Close file... */
00476
          fclose(in);
00477
           /* Check number of points... */
00478
00479
           if (atm->np < 1)
00480
              ERRMSG("Can not read any data!");
00481 }
```

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

Read control parameters.

Definition at line 485 of file libtrac.c.

```
00489
                       {
00490
00491
        int ip, iq;
00492
        /* Write info... */
00493
00494
        printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
                "(executable: %s | compiled: %s, %s)\n\n",
00495
00496
                argv[0], __DATE__, __TIME__);
00497
00498
        /* Initialize quantity indices... */
00499
        ctl->qnt_ens = -1;
ctl->qnt_m = -1;
00500
00501
        ctl->qnt_r = -1;
00502
        ctl->qnt_rho = -1;
00503
        ctl->qnt_ps = -1;
        ctl->qnt_p = -1;
00504
        ctl->qnt_t = -1;
00505
        ctl \rightarrow qnt_u = -1;
00506
00507
        ctl->qnt_v = -1;
        ctl->qnt_w = -1;
00508
        ctl->qnt_h2o = -1;
ctl->qnt_o3 = -1;
00509
00510
        ctl->qnt_theta = -1;
00511
00512
        ctl->qnt\_pv = -1;
00513
        ctl->qnt\_tice = -1;
        ctl->qnt_tsts = -1;
00514
00515
        ctl->qnt_tnat = -1;
00516
        ctl->qnt_gw_var = -1;
00517
        ctl->qnt\_stat = -1;
00518
        /* Read quantities... */
00519
        ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
00520
00521
        if (ctl->nq > NQ)
00522
          ERRMSG("Too many quantities!");
00523
        for (iq = 0; iq < ctl->nq; iq++) {
00524
00525
          /* 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",
00526
```

```
ctl->qnt_format[iq]);
00529
00530
            /* Try to identify quantity... */
            if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
00531
00532
              ctl->qnt_ens = iq;
              sprintf(ctl->qnt_unit[iq], "-");
00533
            } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
00535
              ctl->qnt_m = iq;
00536
              sprintf(ctl->qnt_unit[iq], "kg");
            } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
  ctl->qnt_r = iq;
00537
00538
              sprintf(ctl->qnt_unit[iq], "m");
00539
            } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
  ctl->qnt_rho = iq;
00540
00541
00542
              sprintf(ctl->qnt_unit[iq], "kg/m^3");
00543
            } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
             ctl->qnt_ps = iq;
00544
              sprintf(ctl->qnt_unit[iq], "hPa");
00545
            } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
             ctl->qnt_p = iq;
00547
00548
              sprintf(ctl->qnt_unit[iq], "hPa");
           } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
  ctl->qnt_t = iq;
  sprintf(ctl->qnt_unit[iq], "K");
00549
00550
00551
           } else if (stromp(ctl->qnt_name[iq], "u") == 0) {
ctl->qnt_u = iq;
00552
00554
              sprintf(ctl->qnt_unit[iq], "m/s");
00555
           } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
              ctl->qnt_v = iq;
00556
              sprintf(ctl->qnt_unit[iq], "m/s");
00557
            } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
00558
              ctl->qnt_w = iq;
00559
00560
              sprintf(ctl->qnt_unit[iq], "hPa/s");
00561
            } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
              ctl->qnt_h2o = iq;
00562
              sprintf(ctl->qnt_unit[iq], "1");
00563
           } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
ctl->qnt_o3 = iq;
00564
00566
              sprintf(ctl->qnt_unit[iq], "1");
00567
            } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
00568
             ctl->qnt_theta = iq;
             sprintf(ctl->qnt_unit[iq], "K");
00569
           } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
00570
             ctl->qnt_pv = iq;
sprintf(ctl->qnt_unit[iq], "PVU");
00571
00572
00573
           } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
00574
            ctl->qnt_tice = iq;
00575
              sprintf(ctl->qnt_unit[iq], "K");
           } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
00576
00577
             ctl->qnt_tsts = iq;
              sprintf(ctl->qnt_unit[iq], "K");
00578
00579
            } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
              ctl->qnt_tnat = iq;
sprintf(ctl->qnt_unit[iq], "K");
00580
00581
           } else if (strcmp(ctl->qnt_name[iq], "gw_var") == 0) {
  ctl->qnt_gw_var = iq;
  sprintf(ctl->qnt_unit[iq], "K^2");
00582
00583
00585
            } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
00586
              ctl->qnt_stat = iq;
00587
              sprintf(ctl->qnt_unit[iq], "-");
00588
           1 else
              scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00589
00590
00591
00592
          /★ Time steps of simulation... ★/
00593
         ctl->direction =
         (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
if (ctl->direction != -1 && ctl->direction != 1)
00594
00595
           ERRMSG("Set DIRECTION to -1 or 1!");
00596
         ctl->t_start
         scan_ctl(filename, argc, argv, "T_START", -1, "-le100", NULL);
ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "-le100", NULL);
ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
00598
00599
00600
00601
00602
         /* Meteorological data...
         ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
00603
         ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
00604
00605
         if (ctl->met_np > EP)
00606
           ERRMSG("Too many levels!");
         for (ip = 0; ip < ctl->met_np; ip++)
  ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
00607
00608
00609
00610
00611
         /* Isosurface parameters... */
00612
         ctl->isosurf
         = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL); scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00613
00614
```

```
00615
00616
         /* Diffusion parameters... */
00617
         ctl->turb_dx_trop
00618
           = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00619
         ctl->turb dx strat
           = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00620
00621
         ctl->turb dz trop
00622
            = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00623
         ctl->turb_dz_strat
00624
           = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00625
        ctl->turb_meso =
           scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00626
00627
00628
        /* Life time of particles... */
        ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL); ctl->tdec_strat =
00629
00630
           scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00631
00632
         /* PSC analysis... ∗/
00633
00634
         ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
         ctl->psc_hno3 =
00635
00636
           scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
00637
00638
        /* Gravity wave analysis... */
         scan_ctl(filename, argc, argv, "GW_BASENAME", -1, "-", ctl->
00639
      gw_basename);
00640
         /* Output of atmospheric data... */
scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
00641
00642
      atm basename);
00643
        scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00644
         ctl->atm_dt_out
00645
           scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00646
         ctl->atm_filter
00647
           (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
00648
         ctl->atm bin =
           (int) scan ctl(filename, argc, argv, "ATM BIN", -1, "0", NULL);
00649
00650
00651
         /* Output of CSI data... */
        scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
00652
      csi_basename);
00653 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",
00654
00655
                   ctl->csi_obsfile);
00656
00657
         ctl->csi_obsmin =
00658
           scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00659
        ctl->csi modmin =
           scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
00660
        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);
00661
00662
00663
00664
         ctl->csi lon0 =
         scan_ctl(filename, argc, argv, "CSI_LONO", -1, "-180", NULL);
ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
00665
00666
00667
         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);
00668
00669
00670
00671
         ctl->csi_ny =
00672
           (int) scan ctl(filename, argc, argv, "CSI NY", -1, "180", NULL);
00673
00674
        /* Output of ensemble data... */
        scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
00675
      ens_basename);
00676
00677
         /\star Output of grid data... \star/
        00678
00679
00680
         scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
      grid_gpfile);
00681
        ctl->grid_dt_out =
00682
           scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
00683
         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);
00684
00685
00686
00687
         ctl->grid_nz =
           (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00688
00689
         ct.1->arid lon0 =
00690
           scan ctl(filename, argc, argv, "GRID LONO", -1, "-180", NULL);
00691
         ctl->grid_lon1
00692
           scan ctl(filename, argc, argv, "GRID LON1", -1, "180", NULL);
00693
         ctl->grid_nx =
00694
           (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00695
         ctl->grid lat0 =
00696
           scan ctl(filename, argc, argv, "GRID LATO", -1, "-90", NULL);
```

```
ctl->grid_lat1 =
00698
           scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
         ctl->grid_ny =
00699
00700
          (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
00701
00702
         /* Output of profile data... */
00703
        scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
00704
                    ctl->prof_basename);
00705 scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
prof_obsfile);
00706 ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
00707 ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
00708
        ctl->prof_nz =
00709
            (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
00710
         ctl->prof_lon0 =
00711
           scan_ctl(filename, argc, argv, "PROF_LONO", -1, "-180", NULL);
00712
         ctl->prof_lon1 :
00713
           scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
         ctl->prof_nx =
00715
           (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00716
         ctl->prof lat0 =
00717
           scan_ctl(filename, argc, argv, "PROF_LATO", -1, "-90", NULL);
00718
         ctl->prof_lat1 =
           scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
00719
00720
        ctl->prof_ny =
00721
           (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00722
00723
        /* Output of station data... */
        scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
00724
00725
                   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);
00726
00727
00728
00729 }
```



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

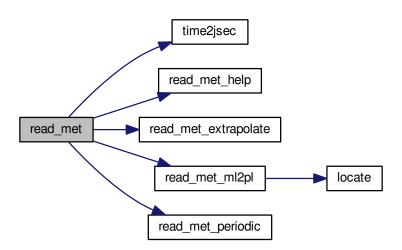
Read meteorological data file.

Definition at line 733 of file libtrac.c.

```
00736
00737
00738
       char cmd[LEN], levname[LEN], tstr[10];
00739
00740
       static float help[EX * EY];
00741
00742
        int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00743
00744
       size_t np, nx, ny;
00745
00746
       /* Write info... */
00747
       printf("Read meteorological data: %s\n", filename);
00748
        /\star Get time from filename... \star/
00749
00750
       sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00751
        vear = atoi(tstr);
00752
       sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00753
        mon = atoi(tstr);
00754
       sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
```

```
day = atoi(tstr);
00756
         sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00757
         hour = atoi(tstr);
00758
         time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00759
00760
         /* Open netCDF file... */
00761
         if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
00762
00763
            /* Try to stage meteo file... */
00764
            STOP TIMER (TIMER INPUT);
00765
            START_TIMER (TIMER_STAGE);
            if (ctl->met_stage[0] != '-') {
00766
              00767
00768
00769
00770
                ERRMSG("Error while staging meteo data!");
00771
00772
            STOP TIMER (TIMER STAGE);
00773
            START_TIMER (TIMER_INPUT);
00774
00775
            /* Try to open again... */
00776
           NC(nc_open(filename, NC_NOWRITE, &ncid));
00777
00778
         /* Get dimensions... */
NC(nc_inq_dimid(ncid, "lon", &dimid));
00779
00780
00781
         NC(nc_inq_dimlen(ncid, dimid, &nx));
00782
         if (nx < 2 \mid \mid nx > EX)
00783
           ERRMSG("Number of longitudes out of range!");
00784
         NC(nc_inq_dimid(ncid, "lat", &dimid));
00785
         NC(nc_inq_dimlen(ncid, dimid, &ny));
00786
00787
             (ny < 2 \mid \mid ny > EY)
00788
            ERRMSG("Number of latitudes out of range!");
00789
         sprintf(levname, "lev");
00790
         NC(nc_inq_dimid(ncid, levname, &dimid));
NC(nc_inq_dimlen(ncid, dimid, &np));
00791
00792
00793
         if (np == 1) {
00794
          sprintf(levname, "lev_2");
00795
           NC(nc_inq_dimid(ncid, levname, &dimid));
00796
           NC(nc_inq_dimlen(ncid, dimid, &np));
00797
00798
         if (np < 2 || np > EP)
00799
            ERRMSG("Number of levels out of range!");
00800
00801
         /* Store dimensions... */
         met->np = (int) np;
met->nx = (int) nx;
00802
00803
         met->ny = (int) ny;
00804
00805
         /* Get horizontal grid... */
NC(nc_inq_varid(ncid, "lon", &varid));
00806
00807
         NC(nc_get_var_double(ncid, varid, met->lon));
NC(nc_inq_varid(ncid, "lat", &varid));
NC(nc_get_var_double(ncid, varid, met->lat));
00808
00809
00810
00811
00812
          /* Read meteorological data... */
         /* Read meteorological data... */
read_met_help(ncid, "t", "T", met, met->t, 1.0);
read_met_help(ncid, "u", "U", met, met->u, 1.0);
read_met_help(ncid, "v", "V", met, met->v, 1.0);
read_met_help(ncid, "w", "W", met, met->w, 0.01f);
read_met_help(ncid, "q", "Q", met, met->help(ncid, "q", "Q", met, met->help(ncid, "o3", "o3", met, met->o3, 0.602f);
00813
00814
00815
00816
00817
00818
00819
00820
          /* Meteo data on pressure levels... */
00821
         if (ctl->met_np <= 0) {
00822
00823
            /* Read pressure levels from file... */
00824
            NC(nc_inq_varid(ncid, levname, &varid));
00825
            NC(nc_get_var_double(ncid, varid, met->p));
            for (ip = 0; ip < met->np; ip++)
met->p[ip] /= 100.;
00826
00827
00828
00829
            /* Extrapolate data for lower boundary... */
00830
            read_met_extrapolate(met);
00831
00832
00833
          /* Meteo data on model levels... */
00834
         else (
00835
00836
            /* Read pressure data from file... */
            read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00837
00838
00839
            /\star Interpolate from model levels to pressure levels... \star/
           read_met_ml2pl(ctl, met, met->t);
read_met_ml2pl(ctl, met, met->u);
00840
00841
```

```
read_met_ml2p1(ct1, met, met->v);
          read_met_ml2pl(ctl, met, met->w);
00844
          read_met_ml2pl(ctl, met, met->h2o);
00845
          read_met_ml2pl(ctl, met, met->o3);
00846
00847
          /\star Set pressure levels... \star/
          met->np = ctl->met_np;
for (ip = 0; ip < met->np; ip++)
00848
00849
00850
            met->p[ip] = ctl->met_p[ip];
00851
00852
        /\star Check ordering of pressure levels... \star/
00853
        for (ip = 1; ip < met->np; ip++)
  if (met->p[ip - 1] < met->p[ip])
00854
00855
00856
            ERRMSG("Pressure levels must be descending!");
00857
        00858
00859
00860
00861
          NC(nc_get_var_float(ncid, varid, help));
          for (iy = 0; iy < met->ny; iy++)
  for (ix = 0; ix < met->nx; ix++)
00862
00863
       00864
00865
00866
          NC(nc_get_var_float(ncid, varid, help));
00868
          for (iy = 0; iy < met->ny; iy++)
00869
            for (ix = 0; ix < met->nx; ix++)
00870
              met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00871
       } else
        for (ix = 0; ix < met->nx; ix++)
  for (iy = 0; iy < met->ny; iy++)
00872
00873
00874
              met \rightarrow ps[ix][iy] = met \rightarrow p[0];
00875
00876
        /\star Create periodic boundary conditions... \star/
00877
       read_met_periodic(met);
00878
        /* Close file... */
08800
       NC(nc_close(ncid));
00881 }
```



5.15.2.20 void read\_met\_extrapolate ( met t \* met )

Extrapolate meteorological data at lower boundary.

Definition at line 885 of file libtrac.c.

```
00887
00888
         int ip, ip0, ix, iy;
00889
00890
         /* Loop over columns... */
00891
         for (ix = 0; ix < met->nx; ix++)
            for (iy = 0; iy < met->ny; iy++) {
00893
00894
               /\star Find lowest valid data point... \star/
               for (ip0 = met->np - 1; ip0 >= 0; ip0--)
   if (!gsl_finite(met->t[ix][iy][ip0])
00895
00896
00897
                      | | !gsl_finite(met->u[ix][iy][ip0])
                      || !gsl_finite(met->v[ix][iy][ip0])
00898
00899
                      || !gsl_finite(met->w[ix][iy][ip0]))
00900
                   break;
00901
00902
               /* Extrapolate... */
              /* Extrapolate... */
for (ip = ip0; ip >= 0; ip--) {
    met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
00903
00904
                 met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
00905
00906
                 met \rightarrow v[ix][iy][ip] = met \rightarrow v[ix][iy][ip + 1];
00907
                 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];
00908
00909
00910
00911
00912 }
```

5.15.2.21 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 916 of file libtrac.c.

```
00923
00924
        static float help[EX * EY * EP];
00925
00926
        int ip, ix, iv, n = 0, varid;
00927
00928
         /* Check if variable exists... */
00929
         if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00930
          if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00931
             return:
00932
00933
         /* Read data... */
00934
         NC(nc_get_var_float(ncid, varid, help));
00935
00936
         /* Copy and check data... */
        for (ip = 0; ip < met->np; ip++)
  for (iy = 0; iy < met->ny; iy++)
00937
00938
             for (ix = 0; ix < met->nx; ix++) {
  dest[ix][iy][ip] = scl * help[n++];
00939
00941
               if (fabs(dest[ix][iy][ip] / scl) > 1e14)
00942
                  dest[ix][iy][ip] = GSL_NAN;
00943
             }
00944 }
```

5.15.2.22 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 948 of file libtrac.c.

```
/* Copy pressure profile... */
              for (ip = 0; ip < met->np; ip++)
  p[ip] = met->pl[ix][iy][ip];
00962
00963
00964
              /* Interpolate... */
for (ip = 0; ip < ctl->met_np; ip++) {
   pt = ctl->met_p[ip];
00965
00966
00967
00968
                if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
00969
                  pt = p[0];
                else if ((pt > p[met->np - 1] && p[1] > p[0])
  || (pt < p[met->np - 1] && p[1] < p[0]))
00970
00971
00972
                  pt = p[met->np - 1];
                00973
00974
00975
00976
00977
00978
              /* Copy data... */
for (ip = 0; ip < ctl->met_np; ip++)
00979
00980
                var[ix][iy][ip] = (float) aux[ip];
00981
00982 }
```



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

Create meteorological data with periodic boundary conditions.

Definition at line 986 of file libtrac.c.

```
00987
00988
00989
          int ip, iy;
00991
          /* Check longitudes... */
00992
          if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
                         + met->lon[1] - met->lon[0] - 360) < 0.01))
00993
00994
            return:
00995
00996
          /* Increase longitude counter... */
00997
          if ((++met->nx) > EX)
00998
             ERRMSG("Cannot create periodic boundary conditions!");
00999
01000
          /* Set longitude... */
         met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
01001
        lon[0];
01002
01003
           /* Loop over latitudes and pressure levels... */
01004
           for (iy = 0; iy < met->ny; iy++)
            for (ip = 0; ip < met->np; ip++) {
    met->ps[met->nx - 1][iy] = met->ps[0][iy];
    met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
    met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
01005
01006
01007
01008
               met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
01009
01010
               met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01011
01012
01013
01014 }
```

5.15.2.24 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 1018 of file libtrac.c.

```
01025
01026
01027
         FILE *in = NULL;
01028
         char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
01029
           msg[LEN], rvarname[LEN], rval[LEN];
01030
01031
01032
         int contain = 0, i;
01033
01034
         /* Open file... */
         if (filename[strlen(filename) - 1] != '-')
if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");
01035
01036
01037
01038
01039
         /* Set full variable name... */
         if (arridx >= 0) {
01040
         sprintf(fullname1, "%s[%d]", varname, arridx);
sprintf(fullname2, "%s[*]", varname);
01041
01042
01043
         sprintf(fullname1, "%s", varname);
sprintf(fullname2, "%s", varname);
01044
01045
01046
01047
01048
         /* Read data... */
         if (in != NULL)
         while (fgets(line, LEN, in))
01050
01051
             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
01052
                if (strcasecmp(rvarname, fullname1) == 0 | |
01053
                    strcasecmp(rvarname, fullname2) == 0) {
01054
                  contain = 1;
01055
                  break;
01056
               }
        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]);
01058
01059
01060
01061
             contain = 1;
01062
             break;
01063
01064
        /* Close file... */
if (in != NULL)
01065
01066
01067
           fclose(in);
01069
         /* Check for missing variables... */
01070
        if (!contain) {
         if (strlen(defvalue) > 0)
01071
             sprintf(rval, "%s", defvalue);
01072
01073
          else {
01074
             sprintf(msg, "Missing variable %s!\n", fullname1);
01075
              ERRMSG(msg);
01076
01077
01078
01079
         /* Write info... */
        printf("%s = %s\n", fullname1, rval);
01080
01082
         /* Return values... */
01083
         if (value != NULL)
01084
          sprintf(value, "%s", rval);
01085
         return atof(rval);
01086 }
```

5.15.2.25 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 1090 of file libtrac.c.

```
01099
01100
       struct tm t0, t1;
01101
       t0.tm_year = 100;
01102
01103
       t0.tm mon = 0;
01104
       t0.tm_mday = 1;
01105
        t0.tm\_hour = 0;
01106
       t0.tm_min = 0;
01107
       t0.tm_sec = 0;
01108
       t1.tm_year = year - 1900;
01109
       t1.tm_mon = mon - 1;
01110
01111
       t1.tm_mday = day;
01112
       t1.tm_hour = hour;
01113
       t1.tm_min = min;
       t1.tm_sec = sec;
01114
01115
01116
       *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01117 }
```

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

Measure wall-clock time.

Definition at line 1121 of file libtrac.c.

```
01124
                   {
01125
01126
        static double starttime[NTIMER], runtime[NTIMER];
01127
       /* Check id... */
if (id < 0 || id >= NTIMER)
01128
01129
01130
          ERRMSG("Too many timers!");
01132
        /* Start timer... */
01133
        if (mode == 1) {
         if (starttime[id] <= 0)</pre>
01134
01135
            starttime[id] = omp_get_wtime();
01136
          else
01137
            ERRMSG("Timer already started!");
01138
01139
        /* Stop timer... */
else if (mode == 2) {
01140
01141
01142
        if (starttime[id] > 0) {
01143
            runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
01144
            starttime[id] = -1;
01145
01146
01147
01148
        /* Print timer... */
01149
        else if (mode == 3)
01150
          printf("%s = %.3f s\n", name, runtime[id]);
01151 }
```

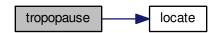
## 5.15.2.27 double tropopause (double t, double lat)

Definition at line 1155 of file libtrac.c.

```
01157
                                     {
01158
              static double doys[12]
01160
              = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01161
01162
              static double lats[73]
                 -8 -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5, -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5, -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5, -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
01163
01164
01165
01166
                 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
01167
01168
01169
01170
01171
```

```
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,
01173
01174
                      175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
01175
                      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,
01176
01177
                                                                                                         113.5,
                      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,
01179
           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,
01180
01181
01182
             150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
01183
             98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
01184
             98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131, 159.5, 193, 220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.
01185
01186
01187
             284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01188
             287.5, 286.2, 285.8},
            {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
01189
             297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
             161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
             100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2, 99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
01192
01193
             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,
01194
01195
            304.3, 304.9, 306.3, 306.6, 306.2, 306), {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4, 290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
01196
01198
01199
             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,
01200
01201
             148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
01202
             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},
01204
01205
            {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,
01206
01207
             101.8, 101.4, 101.1, 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,
01208
01210
             273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01211
01212
             325.3, 325.8, 325.8},
            {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3, 222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2, 228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
01213
01214
             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,
01216
01217
01218
             127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
01219
             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,
01220
01221
             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,
01223
01224
             110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
             111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4, 117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
01225
01226
             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},
01227
            {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01229
             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,
01230
01231
             110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
01232
            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,
01233
             230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
01235
01236
             278.2, 282.6, 287.4, 290.9, 292.5, 293},
01237
            {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
            183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7, 243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5, 114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
01238
01239
             110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
             114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
01242
01243
             203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5
           276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1), {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5, 215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2, 237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
01244
01245
01246
01247
                                 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01248
             111.8, 109.4,
             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,
01249
01250
             206.1, 230.1, 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,
01251
01252
             305.1},
            241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2, 253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
01254
01255
01256
             223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
             108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5, 102.5, 102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
01257
01258
```

```
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}
01260
01261
          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,
01262
01263
01264
01265
01266
01267
            186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
01268
           280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
           281.7, 281.1, 281.2}
01269
01270
          };
01271
01272
          double doy, p0, p1, pt;
01273
01274
          int imon, ilat;
01275
          /* Get day of year... */
doy = fmod(t / 86400., 365.25);
01276
01277
01278
          while (doy < 0)
            doy += 365.25;
01279
01280
01281
          /* Get indices... */
          imon = locate(doys, 12, doy);
ilat = locate(lats, 73, lat);
01282
01283
01284
01285
          /* Get tropopause pressure... */
         01286
01287
          p1 = LIN(lats[ilat], tps[imon + 1][ilat],
lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
01288
01289
01290
          pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01291
01292
          /\star Return tropopause pressure... \star/
          return pt;
01293
01294 }
```



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

Write atmospheric data.

Definition at line 1298 of file libtrac.c.

```
01302
                   {
01303
01304
        FILE *in, *out;
01305
01306
        char line[LEN];
01307
01308
        double r, t0, t1;
01309
01310
        int ip, iq, year, mon, day, hour, min, sec;
01311
01312
        /\star Set time interval for output... \star/
01313
        t0 = t - 0.5 * ctl->dt_mod;
        t1 = t + 0.5 * ctl->dt_mod;
01314
01315
01316
        /\star Check if gnuplot output is requested... \star/
01317
        if (ctl->atm_gpfile[0] != '-') {
01318
```

```
01319
          /* Write info... */
01320
         printf("Plot atmospheric data: %s.png\n", filename);
01321
         /* Create gnuplot pipe... */
if (!(out = popen("gnuplot", "w")))
01322
01323
            ERRMSG("Cannot create pipe to gnuplot!");
01324
01325
          01326
01327
01328
01329
          /* Set time string... */
01330
         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01331
                  year, mon, day, hour, min);
01332
01333
01334
          /* Dump gnuplot file to pipe... */
         if (!(in = fopen(ctl->atm_gpfile, "r")))
    ERRMSG("Cannot open file!");
while (fgets(line, LEN, in))
01335
01336
01337
01338
            fprintf(out, "%s", line);
01339
          fclose(in);
01340
01341
01342
       else {
01343
01344
          /* Write info... */
01345
         printf("Write atmospheric data: %s\n", filename);
01346
01347
          /* Create file... */
         if (!(out = fopen(filename, "w")))
01348
01349
            ERRMSG("Cannot create file!");
01350
01351
01352
        /* Write binary data... */
01353
        if (ctl->atm_bin && ctl->atm_gpfile[0] == '-') {
01354
         FWRITE(&atm->np, int,
01355
                 1,
01356
                 out);
01357
         FWRITE(atm->time, double,
01358
                   (size_t) atm->np,
01359
                 out);
         FWRITE(atm->p, double,
01360
01361
                  (size_t) atm->np,
01362
                 out);
01363
         FWRITE(atm->lon, double,
01364
                   (size_t) atm->np,
01365
                 out);
01366
         FWRITE(atm->lat, double,
01367
                   (size_t) atm->np,
01368
                 out);
          for (iq = 0; iq < ctl->nq; iq++)
01369
01370
            FWRITE(atm->q[iq], double,
01371
                     (size_t) atm->np,
01372
                   out);
01373
01374
01375
       /* Write ASCII data... */
01376
       else {
01377
01378
          /* Write header... */
01379
         fprintf(out,
                   "# $1 = time [s]\n"
01380
01381
                  "# $2 = altitude [km] \n"
01382
                  "# $3 = longitude [deg] \n" "# <math>$4 = latitude [deg] \n");
         01383
01384
01385
01386
01387
01388
          /* Write data... */
01389
          for (ip = 0; ip < atm->np; ip++) {
01390
            /* Check time... */
if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
01391
01392
01393
              continue;
01394
01395
            /* Write output... */
            01396
01397
01398
01399
01400
01401
01402
            fprintf(out, "\n");
01403
         }
01404
01405
```

```
01406  /* Close file... */
01407  fclose(out);
01408 }
```



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

Write CSI data.

Definition at line 1412 of file libtrac.c.

```
01416
01417
01418
        static FILE *in, *out;
01419
01420
        static char line[LEN];
01421
        static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01423
         rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01424
        static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01425
01426
01427
        /* Init... */
01428
        if (!init) {
01429
          init = 1;
01430
          /\star Check quantity index for mass... \star/
01431
         if (ctl->qnt_m < 0)
01432
01433
            ERRMSG("Need quantity mass to analyze CSI!");
01434
01435
          /\star Open observation data file... \star/
          printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
01436
          if (!(in = fopen(ctl->csi_obsfile, "r")))
01437
            ERRMSG("Cannot open file!");
01438
01439
01440
          /* Create new file... */
          printf("Write CSI data: %s\n", filename);
if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");
01441
01442
01443
01444
01445
          /* Write header... */
01446
          fprintf(out,
01447
                   "# $1 = time [s] \n"
01448
                   "# $2 = number of hits (cx) \n"
                   "# $3 = number of misses (cy) \n"
01449
                   "# $4 = number of false alarms (cz)\n"
01450
                   "# $5 = number of observations (cx + cy) \n"
01451
                   "# $6 = number of forecasts (cx + cz) \n"
01452
01453
                   "# \$7 = bias (forecasts/observations) [\%] \n"
                  01454
01455
                  "# $10 = critical success index (CSI) [%%]\n\n");
01456
01457
01458
01459
        /* Set time interval... */
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01460
01461
01462
01463
        /* Initialize grid cells... */
01464
        for (ix = 0; ix < ctl->csi_nx; ix++)
01465
         for (iy = 0; iy < ctl->csi_ny; iy++)
```

```
01466
            for (iz = 0; iz < ctl->csi_nz; iz++)
              modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01467
01468
01469
        /* Read data... */
01470
        while (fgets(line, LEN, in)) {
01471
01472
           /* Read data... *
01473
           if (sscanf(line, "%lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01474
               5)
01475
             continue;
01476
01477
           /* Check time... */
          if (rt < t0)</pre>
01478
            continue;
01479
01480
           if (rt > t1)
01481
            break;
01482
01483
           /* Calculate indices... */
          ix = (int) ((rlon - ctl->csi_lon0))
01484
                        / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01485
01486
           iy = (int) ((rlat - ctl -> csi_lat0))
01487
                        / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
          iz = (int) ((rz - ctl - csi_z0))
01488
                        / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi nz);
01489
01490
01491
           /* Check indices... */
01492
           if (ix < 0 || ix >= ctl->csi_nx ||
01493
               iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
             continue;
01494
01495
01496
           /* Get mean observation index... */
01497
          obsmean[ix][iy][iz] += robs;
01498
          obscount[ix][iy][iz]++;
01499
01500
        /* Analyze model data... */
01501
01502
        for (ip = 0; ip < atm->np; ip++) {
01504
           /* Check time... */
01505
          if (atm->time[ip] < t0 || atm->time[ip] > t1)
01506
             continue;
01507
           /* Get indices... */
01508
          ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01509
                        / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01510
01511
           iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
01512
                        / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
          01513
01514
01515
01516
           /* Check indices... */
01517
          if (ix < 0 || ix >= ctl->csi_nx ||
01518
               iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01519
             continue;
01520
01521
           /* Get total mass in grid cell... */
          modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01522
01523
01524
01525
         /* Analyze all grid cells... */
        for (ix = 0; ix < ctl->csi_nx; ix++)
  for (iy = 0; iy < ctl->csi_ny; iy++)
    for (iz = 0; iz < ctl->csi_nz; iz++) {
01526
01527
01529
01530
               /* Calculate mean observation index... */
               if (obscount[ix][iy][iz] > 0)
  obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01531
01532
01533
01534
               /* 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;
01536
01537
                 lat = ctl->csi_lat0 + dlat * (iy + 0.5);
area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
  * cos(lat * M_PI / 180.);
01538
01539
01540
01541
                 modmean[ix][iy][iz] /= (1e6 * area);
01542
01543
               /* Calculate CSI... */
01544
               if (obscount[ix][iy][iz] > 0) {
01545
                if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01546
01547
                     modmean[ix][iy][iz] >= ctl->csi_modmin)
01548
01549
                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01550
                           modmean[ix][iy][iz] < ctl->csi_modmin)
01551
                   cy++;
01552
                 else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
```

```
modmean[ix][iy][iz] >= ctl->csi_modmin)
01554
01555
01556
01557
        /* Write output... */
if (fmod(t, ctl->csi_dt_out) == 0) {
01558
01560
          01561
01562
                  (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,

(cx + cz > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
01563
01564
01565
01566
01567
                   (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01568
01569
          /* Set counters to zero... */
01570
          cx = cy = cz = 0;
01571
01572
01573
        /* Close file... */
01574
        if (t == ctl->t_stop)
          fclose(out);
01575
01576 }
```

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

Write ensemble data.

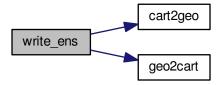
Definition at line 1580 of file libtrac.c.

```
01584
                  {
01585
       static FILE *out:
01586
01587
       static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
         t0, t1, x[NENS][3], xm[3];
01590
01591
       static int init, ip, iq;
01592
01593
       static size t i, n:
01594
01595
        /* Init... */
01596
        if (!init) {
01597
         init = 1;
01598
01599
          /* Check quantities... */
01600
         if (ctl->qnt_ens < 0)</pre>
01601
            ERRMSG("Missing ensemble IDs!");
01602
01603
          /* Create new file... */
         printf("Write ensemble data: %s\n", filename);
if (!(out = fopen(filename, "w")))
01604
01605
01606
            ERRMSG("Cannot create file!");
01607
          /* Write header... */
01608
01609
          fprintf(out,
01610
                  "# $1 = time [s] \n"
                 # $1 - thme [s]\n"

# $2 = altitude [km]\n"

"# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01611
01612
          01613
01614
01615
          01616
01617
01618
          fprintf(out, "# \$%d = number of members\n\n", 5 + 2 * ctl->nq);
01619
01620
01621
       /* Set time interval... */
t0 = t - 0.5 * ctl->dt_mod;
01622
01623
        t1 = t + 0.5 * ct1 - dt_mod;
01624
01625
01626
        /* Init... */
01627
        ens = GSL_NAN;
01628
       n = 0;
01629
01630
        /* Loop over air parcels... */
01631
        for (ip = 0; ip < atm->np; ip++) {
01632
```

```
/* Check time... */
01634
            if (atm->time[ip] < t0 || atm->time[ip] > t1)
01635
                continue;
01636
            /* Check ensemble id... */
if (atm->q[ctl->qnt_ens][ip] != ens) {
01637
01638
01639
01640
                /* Write results... */
01641
                if (n > 0) {
01642
01643
                  /* Get mean position... */
                  xm[0] = xm[1] = xm[2] = 0;
for (i = 0; i < n; i++) {
  xm[0] += x[i][0] / (double) n;
  xm[1] += x[i][1] / (double) n;</pre>
01644
01645
01646
01647
                     xm[2] += x[i][2] / (double) n;
01648
01649
                  cart2geo(xm, &dummy, &lon, &lat);
fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
01650
01651
01652
                             lat);
01653
01654
                  /* Get quantity statistics... */
                  for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01655
01656
01657
01658
                  for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01659
01660
01661
01662
01663
                  fprintf(out, " %lu\n", n);
01664
01665
01666
                /* Init new ensemble... */
               ens = atm->q[ctl->qnt_ens][ip];
n = 0;
01667
01668
             }
01669
01670
01671
             /* Save data...
01672
             p[n] = atm->p[ip];
01673
             geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
            for (iq = 0; iq < ctl->nq; iq++)
  q[iq][n] = atm->q[iq][ip];
if ((++n) >= NENS)
01674
01675
01676
01677
               ERRMSG("Too many data points!");
01678
01679
          /* Write results... */
01680
          if (n > 0) {
01681
01682
01683
             /* Get mean position... */
01684
             xm[0] = xm[1] = xm[2] = 0;
             for (i = 0; i < n; i++) {
    xm[0] += x[i][0] / (double) n;
    xm[1] += x[i][1] / (double) n;
    xm[2] += x[i][2] / (double) n;
01685
01686
01687
01688
01689
01690
             cart2geo(xm, &dummy, &lon, &lat);
             fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
01691
01692
01693
             /* Get quantity statistics... */
             for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
01694
01695
01696
                fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01697
             for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01698
01699
01700
01701
01702
             fprintf(out, " %lu\n", n);
01703
01704
01705
           /\star Close file... \star/
          if (t == ctl->t_stop)
01706
01707
             fclose(out);
01708 }
```



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

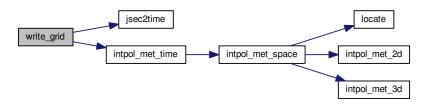
Write gridded data.

Definition at line 1712 of file libtrac.c.

```
01718
                       {
01720
         FILE *in, *out;
01721
01722
         char line[LEN];
01723
01724
         static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01725
            area, rho_air, press, temp, cd, mmr, t0, t1, r;
01726
01727
         static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01728
01729
          /* Check dimensions... */
          if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
01730
01731
            ERRMSG("Grid dimensions too large!");
01732
01733
          /\star Check quantity index for mass... \star/
01734
         if (ctl->qnt_m < 0)</pre>
01735
            ERRMSG("Need quantity mass to write grid data!");
01736
01737
          /* Set time interval for output... */
01738
          t0 = t - 0.5 * ctl->dt_mod;
01739
          t1 = t + 0.5 * ctl -> dt_mod;
01740
         /* Set grid box size... */
dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
01741
01742
01743
01744
         dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
01745
01746
          /* Initialize grid... */
         /* Initialize grid... */
for (ix = 0; ix < ctl->grid_nx; ix++)
    for (iy = 0; iy < ctl->grid_ny; iy++)
    for (iz = 0; iz < ctl->grid_nz; iz++)
01747
01748
01749
01750
                 grid_m[ix][iy][iz] = 0;
01751
01752
          /* Average data... */
         for (ip = 0; ip < atm->np; ip++)
  if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {</pre>
01753
01754
01755
01756
               /* Get index... */
               ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01757
01758
01759
01760
01761
               /* Check indices... */
if (ix < 0 || ix >= ctl->grid_nx ||
01762
01763
                    iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01764
01765
01766
              /* Add mass... */
grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01767
01768
01769
```

```
/* Check if gnuplot output is requested... */
01771
        if (ctl->grid_gpfile[0] != '-') {
01772
01773
           /* Write info... */
01774
           printf("Plot grid data: %s.png\n", filename);
01775
01776
           /* Create gnuplot pipe... */
           if (!(out = popen("gnuplot", "w")))
01777
01778
             ERRMSG("Cannot create pipe to gnuplot!");
01779
           /* Set plot filename... */ fprintf(out, "set out \"%s.png\"\n", filename);
01780
01781
01782
01783
           /* Set time string... */
           jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01784
01785
01786
                    year, mon, day, hour, min);
01787
           /* Dump gnuplot file to pipe... */
           if (!(in = fopen(ctl->grid_gpfile, "r")))
01789
01790
             ERRMSG("Cannot open file!");
           while (fgets(line, LEN, in))
fprintf(out, "%s", line);
01791
01792
01793
           fclose(in);
01794
01795
01796
01797
01798
           /* Write info... */
          printf("Write grid data: %s\n", filename);
01799
01800
01801
           /* Create file... */
01802
           if (!(out = fopen(filename, "w")))
01803
             ERRMSG("Cannot create file!");
01804
01805
         /* Write header... */
01806
         fprintf(out,
01808
                  "# $1 = time [s] \n"
                  "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n"
01809
01810
                  "# $4 = latitude [deg]\n"
01811
                  "# $5 = surface area [km^2]\n"
"# $6 = layer width [km]\n"
01812
01813
                  "# $7 = temperature [K] \n"
01814
01815
                  "# $8 = column density [kg/m^2] n"
01816
                  "# $9 = mass mixing ratio [1]\n\n");
01817
01818
        /* Write data... */
        for (ix = 0; ix < ctl->grid_nx; ix++) {
01819
          if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01821
             fprintf(out, "\n");
           for (iy = 0; iy < ctl->grid_ny; iy++) {
01822
            if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
01823
             fprintf(out, "\n");
for (iz = 0; iz < ctl->grid_nz; iz++)
01824
01825
               if (!ctl->grid_sparse
01827
                    || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01828
01829
                  /* Set coordinates... */
                  z = ctl->grid_z0 + dz * (iz + 0.5);
lon = ctl->grid_lon0 + dlon * (ix + 0.5);
lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01830
01831
01832
01833
01834
                  /* Get pressure and temperature... */
01835
                  press = P(z);
                  intpol_met_time(met0, met1, t, press, lon, lat,
    NULL, &temp, NULL, NULL, NULL, NULL, NULL);
01836
01837
01838
                  /* Calculate surface area... */
                  area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
 * cos(lat * M_PI / 180.);
01840
01841
01842
                  /* Calculate column density... */
01843
                  cd = grid_m[ix][iy][iz] / (1e6 * area);
01844
01845
01846
                  /* Calculate mass mixing ratio... */
                  rho_air = 100. * press / (287.058 * temp);
mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01847
01848
01849
                  01850
01851
01852
01853
                }
01854
          }
01855
01856
```

```
01857  /* Close file... */
01858  fclose(out);
01859 }
```



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

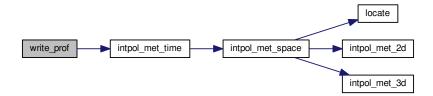
Write profile data.

Definition at line 1863 of file libtrac.c.

```
01869
                      {
01870
01871
         static FILE *in, *out;
01872
01873
         static char line[LEN];
01874
         static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
01875
           rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, mmr, h2o, o3;
01876
01877
01878
01879
         static int init, obscount[GX][GY], ip, ix, iy, iz;
01880
01881
          /* Init... */
         if (!init) {
01882
01883
           init = 1;
01884
01885
            /* Check quantity index for mass... */
01886
           if (ctl->qnt_m < 0)</pre>
              ERRMSG("Need quantity mass!");
01887
01888
           /* Check dimensions... */
if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
    ERRMSG("Grid dimensions too large!");
01889
01890
01891
01892
01893
            /\star Open observation data file... \star/
           printf("Read profile observation data: %s\n", ctl->prof_obsfile);
if (!(in = fopen(ctl->prof_obsfile, "r")))
01894
01895
              ERRMSG("Cannot open file!");
01896
01897
01898
            /* Create new file... */
01899
            printf("Write profile data: %s\n", filename);
            if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");
01900
01901
01902
01903
            /* Write header... */
01904
            fprintf(out,
01905
                      "# $1
                              = time [s]\n"
                      "# $2
                             = altitude [km] \n"
01906
                      "# $3 = longitude [deg] \n"
01907
                      "# $4
                             = latitude [deg]\n"
01908
                             = pressure [hPa]\n"
= temperature [K]\n"
01909
                     "# $5
01910
                     "# $6
01911
                      "# $7 = mass mixing ratio [1]\n"
                      "# $8 = H2O volume mixing ratio [1]\n"
01912
                     "# $9 = 03 volume mixing ratio [1]\n"
"# $10 = mean BT index [K]\n");
01913
01914
01915
01916
            /* 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;
01918
01919
01920
01921
          /* Set time interval... */
01922
         t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
01923
01924
01925
01926
         /* Initialize... */
         for (ix = 0; ix < ctl->prof_nx; ix++)
01927
           for (iy = 0; iy < ctl->prof_ny; iy++) {
01928
              obsmean[ix][iy] = 0;
01929
01930
               obscount[ix][iy] = 0;
01931
               tmean[ix][iy] = 0;
               for (iz = 0; iz < ctl->prof_nz; iz++)
01932
01933
                mass[ix][iy][iz] = 0;
01934
01935
01936
         /* Read data... */
         while (fgets(line, LEN, in)) {
01937
01938
           /* Read data... */
if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01939
01940
01941
              continue;
01942
01943
            /* Check time... */
01944
           if (rt < t0)
           continue;
if (rt > t1)
01945
01946
01947
             break:
01948
01949
            /* Calculate indices... */
           ix = (int) ((rlon - ctl->prof_lon0) / dlon);
iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01950
01951
01952
           /* Check indices... */
if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01953
01954
01955
              continue;
01956
01957
            /\star Get mean observation index... \star/
           obsmean[ix][iy] += robs;
tmean[ix][iy] += rt;
01958
01959
01960
           obscount[ix][iy]++;
01961
01962
01963
         /* Analyze model data... */
01964
         for (ip = 0; ip < atm->np; ip++) {
01965
01966
            /* Check time... */
            if (atm->time[ip] < t0 || atm->time[ip] > t1)
01967
01968
01969
           /* Get indices... */
ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
01970
01971
01972
01973
            iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01974
            /* Check indices... */
01975
           if (ix < 0 || ix >= ctl->prof_nx ||
    iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01976
01977
01978
              continue;
01979
01980
            /\star Get total mass in grid cell... \star/
01981
           mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01982
01983
         /* Extract profiles... */
01984
         for (ix = 0; ix < ctl->prof_nx; ix++)
01985
           for (iy = 0; iy < ctl->prof_ny; iy++)
01987
              if (obscount[ix][iy] > 0) {
01988
                 /* Write output... */
fprintf(out, "\n");
01989
01990
01991
01992
                 /* Loop over altitudes... */
01993
                 for (iz = 0; iz < ctl->prof_nz; iz++) {
01994
                   /* Set coordinates... */
z = ctl->prof_z0 + dz * (iz + 0.5);
lon = ctl->prof_lon0 + dlon * (ix + 0.5);
01995
01996
01997
                   lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01998
01999
02000
                   /* Get meteorological data... */
02001
                   press = P(z);
                   intpol_met_time(met0, met1, t, press, lon, lat,
    NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
02002
02003
```

```
/* 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.);
02005
02006
02007
02008
                   mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
02009
02010
02011
                    /* Write output... */
                   02012
02013
                              z, lon, lat, press, temp, mmr, h2o, o3,
obsmean[ix][iy] / obscount[ix][iy]);
02014
02015
02016
02017
02018
02019
         /* Close file... */
         if (t == ctl->t_stop)
02020
02021
            fclose(out);
02022 }
```



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

Write station data.

Definition at line 2026 of file libtrac.c.

```
02030
02031
       static FILE *out;
02032
02033
       static double rmax2, t0, t1, x0[3], x1[3];
02034
02035
02036
       static int init, ip, iq;
02037
        /* Init... */
02038
        if (!init) {
02039
02040
         init = 1;
02041
02042
          /* Write info... */
02043
         printf("Write station data: %s\n", filename);
02044
         /* Create new file... */
if (!(out = fopen(filename, "w")))
02045
02046
           ERRMSG("Cannot create file!");
02047
02048
02049
          /* Write header... */
02050
         fprintf(out,
                  "# $1 = time [s]\n"
"# $2 = altitude [km]\n"
02051
02052
                  "# $3 = longitude [deg] \n" "# $4 = latitude [deg] \n");
02053
         02054
02055
02056
02057
02058
02059
          /* Set geolocation and search radius... */
02060
         geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
02061
         rmax2 = gsl_pow_2(ctl->stat_r);
```

```
02062
02063
02064
         /\star Set time interval for output... \star/
        t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;
02065
02066
02067
02068
         /* Loop over air parcels... */
02069
         for (ip = 0; ip < atm->np; ip++) {
02070
          /* Check time... */    if (atm->time[ip] < t0 || atm->time[ip] > t1)
02071
02072
02073
            continue:
02074
02075
           /* Check station flag... */
02076
           if (ctl->qnt_stat >= 0)
02077
            if (atm->q[ctl->qnt_stat][ip])
02078
               continue;
02079
02080
           /* Get Cartesian coordinates... */
02081
           geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
02082
02083
           /* Check horizontal distance... */
           if (DIST2(x0, x1) > rmax2)
02084
02085
            continue;
02086
02087
           /* Set station flag... */
02088
           if (ctl->qnt_stat >= 0)
02089
             atm->q[ctl->qnt_stat][ip] = 1;
02090
          /* Write data... */
fprintf(out, "%.2f %g %g %g",
02091
02092
02093
                   atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
           for (iq = 0; iq < ctl->nq; iq++) {
  fprintf(out, " ");
02094
02095
02096
             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02097
02098
          fprintf(out, "\n");
02099
02100
02101
         /* Close file... */
02102
        if (t == ctl->t_stop)
           fclose(out);
02103
02104 }
```

Here is the call graph for this function:



## 5.16 libtrac.h

```
00001 /*
         This file is part of MPTRAC.
00002
00003
00004
          MPTRAC is free software: you can redistribute it and/or modify
00005
          it under the terms of the GNU General Public License as published by
00006
          the Free Software Foundation, either version 3 of the License, or
00007
          (at your option) any later version.
00008
00009
          MPTRAC is distributed in the hope that it will be useful,
          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) 2013-2018 Forschungszentrum Juelich GmbH
```

5.16 libtrac.h 135

```
00018 */
00019
00034 #include <ctype.h>
00035 #include <gsl/gsl_const_mksa.h>
00036 #include <gsl/gsl_math.h>
00037 #include <gsl/gsl_randist.h>
00038 #include <gsl/gsl_rng.h>
00039 #include <gsl/gsl_sort.h>
00040 #include <gsl/gsl_statistics.h>
00041 #include <math.h>
00042 #include <netcdf.h>
00043 #include <omp.h>
00044 #include <stdio.h>
00045 #include <stdlib.h>
00046 #include <string.h>
00047 #include <time.h>
00048 #include <sys/time.h>
00049
00050 /* ---
00051
        Constants...
00052
00053
00055 #define G0 9.80665
00056
00058 #define H0 7.0
00061 #define P0 1013.25
00062
00064 #define RE 6367.421
00065
00066 /*
00067
         Dimensions...
00068
00069
00071 #define LEN 5000
00072
00074 #define NP 10000000
00075
00077 #define NQ 12
00078
00080 #define EP 111
00081
00083 #define EX 1201
00084
00086 #define EY 601
00087
00089 #define GX 720
00090
00092 #define GY 360
00093
00095 #define GZ 100
00096
00098 #define NENS 2000
00099
00101 #define NTHREADS 128
00102
00104
00105
00106
00108 #define ALLOC(ptr, type, n)
00109 if((ptr=calloc((size_t)(n), sizeof(type))) == NULL)
00110
          ERRMSG("Out of memory!");
00111
00113 #define DIST(a, b) sqrt(DIST2(a, b))
00114
00116 #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]))
00117
00118
00120 #define DOTP(a, b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
00121
00123 #define ERRMSG(msg)
00124 printf("\nError (%s, %s, 1%d): %s\n\n",
00125 __FILE__, _func__, _LINE__, msg);
          exit(EXIT_FAILURE);
00126
00127 }
00128
00130 #define FREAD(ptr, type, size, out) {
00131     if(fread(ptr, sizeof(type), size, out)!=size)
00132     ERRMSG("Error while reading!");
00133
00134
00136 #define FWRITE(ptr, type, size, out) {
00137    if(fwrite(ptr, sizeof(type), size, out)!=size)
00138
            ERRMSG("Error while writing!");
00139
00140
```

```
00142 #define LIN(x0, y0, x1, y1, x)
00143
     ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0))
00144
00146 #define NC(cmd) {
       if((cmd)!=NC_NOERR)
00147
00148
           ERRMSG(nc_strerror(cmd));
00149
00150
00152 #define NORM(a) sqrt(DOTP(a, a))
00153
00155 #define PRINT(format, var)
00156 printf("Print (%s, %s, 1%d): %s= "format"\n", 00157 ___FILE__, __func__, __LINE__, #var, var);
00158
00160 #define P(z) (P0*exp(-(z)/H0))
00161
00166
         } else ERRMSG("Error while reading!");
00167
00168
00170 #define Z(p) (H0*log(P0/(p)))
00171
00172 /*
00173
       Timers...
00174
00175
00177 #define START_TIMER(id) timer(#id, id, 1)
00178
00180 #define STOP_TIMER(id) timer(#id, id, 2)
00181
00183 #define PRINT_TIMER(id) timer(#id, id, 3)
00184
00186 #define NTIMER 13
00187
00189 #define TIMER_TOTAL 0
00190
00192 #define TIMER_INIT 1
00193
00195 #define TIMER_STAGE 2
00196
00198 #define TIMER INPUT 3
00199
00201 #define TIMER_OUTPUT 4
00202
00204 #define TIMER_ADVECT 5
00205
00207 #define TIMER_DECAY 6
00208
00210 #define TIMER_DIFFMESO 7
00211
00213 #define TIMER_DIFFTURB 8
00214
00216 #define TIMER ISOSURF 9
00217
00219 #define TIMER_METEO 10
00220
00222 #define TIMER_POSITION 11
00223
00225 #define TIMER SEDI 12
00226
00227 /*
00228
       Structs...
00229
00230
00232 typedef struct {
00233
00235
       int ng:
00236
00238
       char qnt_name[NQ][LEN];
00239
00241
       char qnt_unit[NQ][LEN];
00242
00244
       char gnt format[NO][LEN];
00245
00247
       int qnt_ens;
00248
00250
       int qnt_m;
00251
00253
       int qnt_rho;
00254
00256
       int qnt_r;
00257
00259
       int qnt_ps;
00260
00262
       int qnt_p;
```

5.16 libtrac.h 137

```
00263
00265
        int qnt_t;
00266
00268
        int qnt_u;
00269
00271
        int qnt_v;
00272
00274
        int qnt_w;
00275
00277
        int qnt_h2o;
00278
00280
        int qnt_o3;
00281
00283
        int qnt_theta;
00284
00286
        int qnt_pv;
00287
00289
        int qnt_tice;
00290
00292
        int qnt_tsts;
00293
00295
        int qnt_tnat;
00296
00298
        int qnt_stat;
00299
00301
        int qnt_gw_u750;
00302
00304
        int qnt_gw_v750;
00305
00307
        int qnt_gw_sso;
00308
00310
        int gnt gw var;
00311
00313
        int direction;
00314
00316
        double t_start;
00317
        double t_stop;
00320
00322
        double dt_mod;
00323
00325
        double dt_met;
00326
00328
        int met_np;
00329
00331
        double met_p[EP];
00332
00334
        char met_stage[LEN];
00335
00338
        int isosurf:
00339
00341
        char balloon[LEN];
00342
00344
        double turb_dx_trop;
00345
00347
        double turb_dx_strat;
00348
00350
        double turb_dz_trop;
00351
00353
        double turb_dz_strat;
00354
00356
        double turb_meso;
00357
00359
        double tdec_trop;
00360
00362
        double tdec_strat;
00363
00365
        double psc_h2o;
00366
        double psc_hno3;
00369
00371
        char gw_basename[LEN];
00372
00374
        char atm_basename[LEN];
00375
        char atm_gpfile[LEN];
00378
00380
        double atm_dt_out;
00381
00383
        int atm_filter;
00384
00386
        int atm_bin;
00387
00389
        char csi_basename[LEN];
00390
00392
        double csi_dt_out;
00393
```

```
00395
        char csi_obsfile[LEN];
00396
00398
        double csi_obsmin;
00399
00401
        double csi modmin;
00402
00404
        int csi_nz;
00405
00407
        double csi_z0;
00408
        double csi_z1;
00410
00411
00413
        int csi_nx;
00414
00416
        double csi_lon0;
00417
        double csi_lon1;
00419
00420
00422
        int csi_ny;
00423
00425
        double csi_lat0;
00426
        double csi_lat1;
00428
00429
00431
        char grid_basename[LEN];
00432
00434
        char grid_gpfile[LEN];
00435
00437
        double grid_dt_out;
00438
00440
        int grid_sparse;
00441
00443
        int grid_nz;
0\,0\,4\,4\,4
00446
        double grid_z0;
00447
00449
        double grid_z1;
00450
00452
        int grid_nx;
00453
00455
        double grid_lon0;
00456
        double grid_lon1;
00458
00459
00461
        int grid_ny;
00462
00464
        double grid_lat0;
00465
        double grid_lat1;
00467
00468
        char prof_basename[LEN];
00471
00473
        char prof_obsfile[LEN];
00474
00476
        int prof_nz;
00477
        double prof_z0;
00480
00482
        double prof_z1;
00483
00485
        int prof_nx;
00486
00488
        double prof_lon0;
00489
00491
        double prof_lon1;
00492
00494
        int prof_ny;
00495
00497
        double prof_lat0;
00498
00500
        double prof_lat1;
00501
00503
        char ens_basename[LEN];
00504
00506
        char stat basename[LEN];
00507
00509
        double stat_lon;
00510
00512
        double stat_lat;
00513
00515
        double stat_r;
00516
00517 } ctl_t;
00518
00520 typedef struct {
00521
00523
        int np;
```

5.16 libtrac.h 139

```
00524
00526
        double time[NP];
00527
00529
        double p[NP];
00530
00532
        double lon[NP];
00533
00535
        double lat[NP];
00536
00538
        double q[NQ][NP];
00539
00541
        float up[NP];
00542
00544
        float vp[NP];
00545
00547
00548
        float wp[NP];
00549 } atm_t;
00550
00552 typedef struct {
00553
00555
        double time;
00556
00558
        int nx;
00559
        int ny;
00562
00564
        int np;
00565
00567
        double lon[EX];
00568
        double lat[EY];
00571
00573
        double p[EP];
00574
00576
        double ps[EX][EY];
00577
        float pl[EX][EY][EP];
00580
00582
        float t[EX][EY][EP];
00583
00585
        float u[EX][EY][EP];
00586
00588
        float v[EX][EY][EP];
00589
00591
        float w[EX][EY][EP];
00592
00594
        float h2o[EX][EY][EP];
00595
00597
        float o3[EX][EY][EP];
00598
00599 } met_t;
00600
00601 /* -----
00602
         Functions...
00603
00604
00606 void cart2geo(
00607
        double *x,
00608
        double *z,
00609
        double *lon,
00610
       double *lat);
00611
00613 double deg2dx(
00614
       double dlon,
00615
       double lat);
00616
00618 double deg2dv(
00619
       double dlat);
00620
00622 double dp2dz(
00623
       double dp,
00624
       double p);
00625
00627 double dx2deg(
00628
      double dx,
00629
        double lat);
00630
00632 double dy2deg(
00633
        double dy);
00634
00636 double dz2dp(
00637
       double dz,
00638
        double p);
00639
00641 void geo2cart(
00642
       double z.
```

```
00643
        double lon,
00644
        double lat,
00645
        double *x);
00646
00648 void get_met(
        ctl_t * ctl,
char *metbase,
00649
00650
00651
        double t,
        met_t * met0,
met_t * met1);
00652
00653
00654
00656 void get_met_help(
00657
        double t,
00658
        int direct,
00659
        char *metbase,
        double dt_met,
char *filename);
00660
00661
00662
00664 void intpol_met_2d(
00665
        double array[EX][EY],
00666
        int ix,
00667
        int iy,
00668
        double wx,
00669
        double wy,
00670
        double *var);
00671
00673 void intpol_met_3d(
00674
        float array[EX][EY][EP],
00675
        int ip,
00676
        int ix,
        int iy,
00677
00678
        double wp,
00679
        double wx,
00680
        double wy,
00681
        double *var);
00682
00684 void intpol_met_space(
00685
        met_t * met,
00686
        double p,
00687
        double lon,
00688
        double lat,
00689
        double *ps,
00690
        double *t,
00691
        double *u,
00692
        double *v,
00693
        double *w,
00694
        double *h2o,
00695
        double *o3);
00696
00698 void intpol_met_time(
       met_t * met0,
met_t * met1,
00699
00700
00701
        double ts,
00702
        double p,
00703
        double lon,
00704
        double lat,
00705
        double *ps,
00706
        double *t,
00707
        double *u,
00708
        double *v,
00709
        double *w,
00710
        double *h2o,
00711
        double *o3);
00712
00714 void jsec2time(
00715
        double jsec,
00716
        int *year,
00717
        int *mon.
00718
        int *day,
00719
        int *hour,
00720
        int *min,
00721
        int *sec,
00722
        double *remain);
00723
00725 int locate(
00726
       double *xx,
00727
        int n,
00728
        double x);
00729
00731 void read atm(
00732
       const char *filename,
ctl_t * ctl,
00733
00734
        atm_t * atm);
00735
00737 void read_ctl(
00738 const char *filename,
00739
       int argc,
```

5.16 libtrac.h 141

```
00740
        char *argv[],
00741
        ctl_t * ctl);
00742
00744 void read_met(
00745
       ctl_t * ctl,
char *filename,
00746
00747
        met_t * met);
00748
00750 void read_met_extrapolate(
00751
       met_t * met);
00752
00754 void read_met_help(
00755
        int ncid,
00756
        char *varname,
00757
        char *varname2,
00758
        met_t * met,
00759
        float dest[EX][EY][EP],
00760
        float scl);
00761
00763 void read_met_ml2pl(
      ctl_t * ctl,
met_t * met,
00764
00765
00766
       float var[EX][EY][EP]);
00767
00769 void read_met_periodic(
00770
       met_t * met);
00771
00773 double scan_ctl(
00774
        const char *filename,
00775
        int argc,
char *argv[],
00776
00777
        const char *varname,
00778
        int arridx,
00779
        const char *defvalue,
00780
        char *value);
00781
00783 void time2jsec(
00784
       int year,
00785
        int mon,
00786
        int day,
00787
        int hour,
00788
        int min,
00789
        int sec,
double remain,
00790
00791
        double *jsec);
00792
00794 void timer(
00795 const char *name,
00796
        int id,
00797
        int mode);
00798
00799 /* Get tropopause pressure... */
00800 double tropopause(
00801
        double t,
00802
        double lat);
00803
00805 void write_atm(
00806
       const char *filename,
        ctl_t * ctl,
atm_t * atm,
00807
00808
00809
        double t);
00810
00812 void write_csi(
00813
       const char *filename,
00814
        ctl_t * ctl,
        atm_t * atm,
00815
00816
        double t);
00817
00819 void write_ens(
       const char *filename,
        ctl_t * ctl,
atm_t * atm,
00821
00822
00823
        double t);
00824
00826 void write_grid(
00827
       const char *filename,
00828
        ctl_t * ctl,
00829
        met_t * met0,
        met_t * met1,
atm_t * atm,
00830
00831
00832
        double t);
00833
00835 void write_prof(
00836
        const char *filename,
        ctl_t * ctl,
met_t * met0,
met_t * met1,
00837
00838
00839
```

```
00840 atm_t * atm,

00841 double t);

00842

00844 void write_station(

00845 const char *filename,

00846 ctl_t * ctl,

00847 atm_t * atm,

00848 double t);
```

### 5.17 match.c File Reference

Calculate deviations between two trajectories.

### **Functions**

• int main (int argc, char \*argv[])

### 5.17.1 Detailed Description

Calculate deviations between two trajectories.

Definition in file match.c.

#### 5.17.2 Function Documentation

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

Definition at line 27 of file match.c.

```
00029
00030
00031
         ctl t ctl;
00032
00033
         atm_t *atm1, *atm2, *atm3;
00034
00035
         FILE *out;
00036
00037
         char filename[LEN];
00038
00039
         double filter_dt, x1[3], x2[3], dh, dq[NQ], dv, 1h = 0, 1t = 0, 1v = 0;
00040
00041
         int filter, ip1, ip2, iq, n;
00042
00043
         /* Allocate... */
         ALLOC(atm1, atm_t, 1);
ALLOC(atm2, atm_t, 1);
00044
00045
00046
         ALLOC(atm3, atm_t, 1);
00047
         /* Check arguments... */
if (argc < 5)</pre>
00048
00049
00050
           ERRMSG("Give parameters: <ctl> <atm_test> <atm_ref> <outfile>");
00051
00052
         /* Read control parameters... */
         read_ctl(argv[1], argc, argv, &ctl);
filter = (int) scan_ctl(argv[1], argc, argv, "FILTER", -1, "0", NULL);
filter_dt = scan_ctl(argv[1], argc, argv, "FILTER_DT", -1, "0", NULL);
00053
00054
00055
00056
00057
         /* Read atmospheric data... */
00058
         read_atm(argv[2], &ctl, atm1);
00059
         read_atm(argv[3], &ctl, atm2);
00060
00061
         /* Write info... */
00062
         printf("Write transport deviations: %s\n", argv[4]);
00063
00064
         /* Create output file... */
```

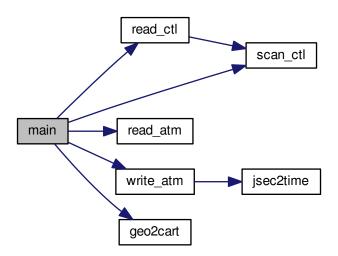
```
if (!(out = fopen(argv[4], "w")))
00066
           ERRMSG("Cannot create file!");
00067
00068
         /* Write header... */
00069
         00070
                   "# $2 = altitude [km] \n"
00072
                   "# $3 = longitude [deg]\n" "# <math>$4 = latitude [deg]\n");
         for (iq = 0; iq < ctl.nq; iq++)
  fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],</pre>
00073
00074
00075
                     ctl.qnt_unit[iq]);
00076
         fprintf(out,
                   "# $%d = trajectory time [s]\n"
"# $%d = vertical length of trajectory [km]\n"
00077
00078
00079
                   "# \$%d = horizontal length of trajectory [km]\n"
00080
                   "# \$%d = vertical deviation [km]\n"
                   "# $%d = horizontal deviation [km]\n",
00081
         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,
00082
00083
00084
         ctl.qnt_name[iq], ctl.qnt_unit[iq]);
fprintf(out, "\n");
00085
00086
00087
00088
         /\star Filtering of reference time series... \star/
00089
         if (filter) {
00090
00091
            /* Copy data... */
00092
           memcpy(atm3, atm2, sizeof(atm_t));
00093
00094
            /* Loop over data points... */
00095
           for (ip1 = 0; ip1 < atm2->np; ip1++) {
00096
             n = 0;
00097
              atm2->p[ip1] = 0;
00098
              for (iq = 0; iq < ctl.nq; iq++)</pre>
              atm2->q[iq][ip1] = 0;
for (ip2 = 0; ip2 < atm2->np; ip2++)
00099
00100
                if (fabs(atm2->time[ip1] - atm2->time[ip2]) < filter_dt) {
    atm2->p[ip1] += atm3->p[ip2];
00101
00102
00103
                   for (iq = 0; iq < ctl.nq; iq++)</pre>
00104
                     atm2->q[iq][ip1] += atm3->q[iq][ip2];
00105
                   n++;
00106
              atm2->p[ip1] /= n;
for (iq = 0; iq < ctl.nq; iq++)
00107
00108
                atm2->q[iq][ip1] /= n;
00109
00110
00111
           /* Write filtered data... */
sprintf(filename, "%s.filt", argv[3]);
00112
00113
           write_atm(filename, &ctl, atm2, 0);
00114
00115
00116
00117
          /* Loop over air parcels (reference data)... */
00118
         for (ip2 = 0; ip2 < atm2->np; ip2++) {
00119
00120
            /* Get trajectory length... */
           if (ip2 > 0) {
00121
00122
              geo2cart(0, atm2->lon[ip2 - 1], atm2->lat[ip2 - 1], x1);
00123
              geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00124
              lh += DIST(x1, x2);
              lv += fabs(Z(atm2->p[ip2 - 1]) - Z(atm2->p[ip2]));
lt = fabs(atm2->time[ip2] - atm2->time[0]);
00125
00126
00127
00128
00129
            /* Init... */
00130
           n = 0;
00131
            dh = 0;
00132
            dv = 0;
           for (iq = 0; iq < ctl.nq; iq++)
00133
             dq[iq] = 0;
00134
00135
            geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00136
           /* Find corresponding time step (test data)... */
for (ip1 = 0; ip1 < atm1->np; ip1++)
  if (fabs(atm1->time[ip1] - atm2->time[ip2])
00137
00138
00139
                   < (filter ? filter_dt : 0.1)) {
00140
00141
00142
                / \star \ \texttt{Calculate deviations...} \ \star /
00143
                geo2cart(0, atm1->lon[ip1], atm1->lat[ip1], x1);
                dh += DIST(x1, x2);
00144
                dv += Z(atm1->p[ip1]) - Z(atm2->p[ip2]);
00145
00146
                for (iq = 0; iq < ctl.nq; iq++)</pre>
00147
                   dq[iq] += atm1->q[iq][ip1] - atm2->q[iq][ip2];
00148
                n++;
00149
              }
00150
00151
            /* Write output... */
```

```
if (n > 0) {
00153
                fprintf(out, "%.2f %.4f %.4f %.4f",
                      atm2->time[ip2], Z(atm2->p[ip2]),
atm2->lon[ip2], atm2->lat[ip2]);
00154
00155
                 for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");
  fprintf(out, ctl.qnt_format[iq], atm2->q[iq][ip2]);
00156
00157
00158
00159
                fprintf(out, " %.2f %g %g %g %g", lt, lv, lh, dv / n, dh / n);
for (iq = 0; iq < ctl.nq; iq++) {
    fprintf(out, " ");
    fprintf(out, ctl.qnt_format[iq], dq[iq] / n);</pre>
00160
00161
00162
00163
00164
00165
                 fprintf(out, "\n");
00166
00167
00168
           /* Close file... */
00169
00170
          fclose(out);
00171
00172
           /* Free... */
00173
          free(atm1);
00174
          free (atm2);
00175
           free (atm3);
00176
           return EXIT_SUCCESS;
00178 }
```

Here is the call graph for this function:



# 5.18 match.c

```
00001 /*
00002
        This file is part of MPTRAC.
00003
00004
        MPTRAC is free software: you can redistribute it and/or modify
        it under the terms of the GNU General Public License as published by
00005
        the Free Software Foundation, either version 3 of the License, or
00006
00007
        (at your option) any later version.
80000
00009
        MPTRAC is distributed in the hope that it will be useful,
        but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
00012
        GNU General Public License for more details.
00013
00014
        You should have received a copy of the GNU General Public License
```

5.18 match.c 145

```
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
        atm_t *atm1, *atm2, *atm3;
00034
00035
        FILE *out;
00036
00037
        char filename[LEN];
00038
00039
         double filter_dt, x1[3], x2[3], dh, dq[NQ], dv, 1h = 0, 1t = 0, 1v = 0;
00040
00041
         int filter, ip1, ip2, iq, n;
00042
         /* Allocate... */
00043
        ALLOC(atm1, atm_t, 1);
ALLOC(atm2, atm_t, 1);
00044
00045
00046
         ALLOC(atm3, atm_t, 1);
00047
00048
         /* Check arguments... */
00049
         if (argc < 5)
00050
           ERRMSG("Give parameters: <ctl> <atm test> <atm ref> <outfile>");
00051
00052
         /* Read control parameters... */
00053
         read_ctl(argv[1], argc, argv, &ctl);
         filter = (int) scan_ctl(argv[1], argc, argv, "FILTER", -1, "0", NULL);
filter_dt = scan_ctl(argv[1], argc, argv, "FILTER_DT", -1, "0", NULL);
00054
00055
00056
00057
         /* Read atmospheric data... */
00058
         read_atm(argv[2], &ctl, atml);
00059
         read_atm(argv[3], &ctl, atm2);
00060
00061
         /* Write info... */
00062
         printf("Write transport deviations: %s\n", argv[4]);
00063
00064
         /* Create output file...
         if (!(out = fopen(argv[4], "w")))
00065
00066
           ERRMSG("Cannot create file!");
00067
00068
         /* Write header... */
00069
         fprintf(out.
                   "# $1 = time [s] \n"
00070
00071
                  "# $2 = altitude [km] \n"
                  "# $3 = longitude [deg] \n" "# <math>$4 = latitude [deg] \n");
00072
         for (iq = 0; iq < ctl.nq; iq++)
  fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],</pre>
00073
00074
00075
                    ctl.qnt_unit[iq]);
00076
         fprintf(out,
00077
                  "# $%d = trajectory time [s]\n"
                  "# \$%d = vertical length of trajectory [km]\n"
00078
                   "# \$%d = horizontal length of trajectory [km]\n"
00079
                  "# \$%d = vertical deviation [km]\n"
08000
                  "# \$%d = horizontal deviation [km]\n",
00081
00082
                  5 + ctl.nq, 6 + ctl.nq, 7 + ctl.nq, 8 + ctl.nq, 9 + ctl.nq);
         for (iq = 0; iq < ctl.nq; iq+)
fprintf(out, "# $%d = %s deviation [%s]\n", ctl.nq + iq + 10,
00083
00084
        ctl.qnt_name[iq], ctl.qnt_unit[iq]);
fprintf(out, "\n");
00085
00086
00087
00088
         /\star Filtering of reference time series... \star/
00089
         if (filter) {
00090
00091
           /* Copy data... */
00092
           memcpy(atm3, atm2, sizeof(atm_t));
00093
           /* Loop over data points... */
for (ip1 = 0; ip1 < atm2->np; ip1++) {
00094
00095
00096
00097
              atm2->p[ip1] = 0;
00098
              for (iq = 0; iq < ctl.nq; iq++)</pre>
00099
               atm2->q[iq][ip1] = 0;
              atm2->q[tq][tp1] - 0,
for (ip2 = 0; ip2 < atm2->np; ip2++)
    if (fabs(atm2->time[ip1] - atm2->time[ip2]) < filter_dt) {
00100
00101
                 atm2->p[ip1] += atm3->p[ip2];
00102
00103
                  for (iq = 0; iq < ctl.nq; iq++)</pre>
00104
                    atm2->q[iq][ip1] += atm3->q[iq][ip2];
00105
                  n++;
00106
                }
```

```
atm2->p[ip1] /= n;
00108
             for (iq = 0; iq < ctl.nq; iq++)</pre>
00109
               atm2->q[iq][ip1] /= n;
00110
00111
           /* Write filtered data... */
00112
           sprintf(filename, "%s.filt", argv[3]);
00113
00114
           write_atm(filename, &ctl, atm2, 0);
00115
00116
         /\star Loop over air parcels (reference data)... \star/
00117
         for (ip2 = 0; ip2 < atm2->np; ip2++) {
00118
00119
00120
           /* Get trajectory length... */
00121
           if (ip2 > 0) {
00122
             geo2cart(0, atm2->lon[ip2 - 1], atm2->lat[ip2 - 1], x1);
              geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00123
             lh += DIST(x1, x2);
lv += fabs(Z(atm2->p[ip2 - 1]) - Z(atm2->p[ip2]));
00124
00126
             lt = fabs(atm2->time[ip2] - atm2->time[0]);
00127
00128
           /* Init... */
00129
           n = 0;
00130
00131
           dh = 0;
00132
           dv = 0;
00133
           for (iq = 0; iq < ctl.nq; iq++)</pre>
00134
             dq[iq] = 0;
00135
           geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00136
00137
           /\star Find corresponding time step (test data)... \star/
           for (ipl = 0; ipl < atml->np; ipl++)
if (fabs(atml->time[ipl] - atm2->time[ip2])
00138
00139
00140
                  < (filter ? filter_dt : 0.1)) {
00141
               /* Calculate deviations... */
00142
00143
               geo2cart(0, atm1->lon[ip1], atm1->lat[ip1], x1);
                dh += DIST(x1, x2);
00145
                dv += Z(atm1->p[ip1]) - Z(atm2->p[ip2]);
00146
               for (iq = 0; iq < ctl.nq; iq++)</pre>
00147
                 dq[iq] += atm1->q[iq][ip1] - atm2->q[iq][ip2];
00148
               n++;
00149
00150
00151
           /* Write output... */
00152
           if (n > 0) {
00153
             fprintf(out, "%.2f %.4f %.4f %.4f",
                      atm2->time[ip2], Z(atm2->p[ip2]),
00154
                      atm2->lon[ip2], atm2->lat[ip2]);
00155
             for (iq = 0; iq < ctl.nq; iq++) {
  fprintf(out, " ");</pre>
00156
00157
00158
                fprintf(out, ctl.qnt_format[iq], atm2->q[iq][ip2]);
00159
             fprintf(out, " %.2f %g %g %g %g", lt, lv, lh, dv / n, dh / n);
for (iq = 0; iq < ctl.nq; iq++) {
   fprintf(out, " ");
   fprintf(out, ctl.qnt_format[iq], dq[iq] / n);</pre>
00160
00161
00162
00164
              fprintf(out, "\n");
00165
00166
           }
00167
        }
00168
00169
         /* Close file... */
00170
        fclose(out);
00171
00172
         /* Free... */
00173
        free(atm1);
00174
        free(atm2);
00175
        free(atm3):
00177
         return EXIT_SUCCESS;
00178 }
```

## 5.19 met\_map.c File Reference

Extract global map from meteorological data.

## **Functions**

• int main (int argc, char \*argv[])

## 5.19.1 Detailed Description

Extract global map from meteorological data.

Definition in file met map.c.

### 5.19.2 Function Documentation

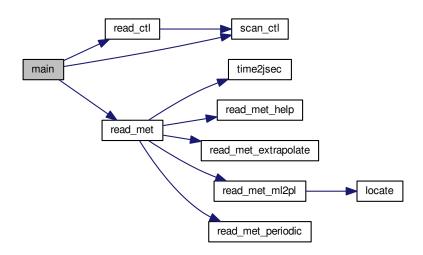
```
5.19.2.1 int main ( int argc, char * argv[])
```

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

```
00029
00030
00031
         ctl_t ctl;
00032
00033
         met_t *met;
00034
00035
        FILE *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
         /\star Check arguments... \star/
00046
         if (argc < 4)
           ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00047
00048
00049
        /* Read control parameters... */
        read_ct1(argv[1], argc, argv, &ct1);
z = scan_ct1(argv[1], argc, argv, "Z", -1, "", NULL);
00050
00051
00052
00053
         /* Loop over files... */
00054
         for (i = 3; i < argc; i++) {</pre>
00055
00056
            /\star Read meteorological data... \star/
00057
           read_met(&ctl, argv[i], met);
00058
00059
           /* Find nearest pressure level... */
           for (ip2 = 0; ip2 < met->np; ip2++) {
    dz = fabs(Z(met->p[ip2]) - z);
00060
00061
00062
              if (dz < dzmin) {
00063
                dzmin = dz;
00064
                ip = ip2;
00065
             }
00066
           }
00067
00068
           /* Average data... */
           for (ix = 0; ix < met->nx; ix++)
  for (iy = 0; iy < met->ny; iy++) {
00069
00070
               timem[ix][iy] += met->time;
tm[ix][iy] += met->t[ix][iy][ip];
um[ix][iy] += met->u[ix][iy][ip];
00071
00072
00073
00074
                vm[ix][iy] += met->v[ix][iy][ip];
00075
                wm[ix][iy] += met->w[ix][iy][ip];
00076
                h2om[ix][iy] += met->h2o[ix][iy][ip];
                o3m[ix][iy] += met->o3[ix][iy][ip];
psm[ix][iy] += met->ps[ix][iy];
00077
00078
00079
                np[ix][iy]++;
08000
00081
00082
00083
        /* Create output file... */
         printf("Write meteorological data file: sn'', argv[2]);
00084
         if (!(out = fopen(argv[2], "w")))
00085
           ERRMSG("Cannot create file!");
00086
00087
         /* Write header... */
00088
        fprintf(out, "# $1
00089
00090
                          = time [s]\n"
                  "# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n"
00091
00092
                  "# $4 = latitude [deg]\n"
00093
```

```
"# $5 = pressure [hPa] \n'
00095
             "# $6 = temperature [K]\n"
             "# $7 = zonal wind [m/s] n"
00096
             "# $8 = meridional wind [m/s]\n"
00097
             "# $9 = vertical wind [hPa/s] \n"
00098
             "# $10 = H20 volume mixing ratio [1]\n"
"# $11 = 03 volume mixing ratio [1]\n"
00099
00100
00101
             "# $12 = surface pressure [hPa] \n");
00102
      00103
00104
00105
00106
00107
00108
                  00109
00110
00111
00112
00113
00114
00115
        for (ix = 0; ix < met\rightarrownx; ix++)
         00116
00117
00118
00119
00120
00121
00122
00123
00124
00125
00126
       /* Close file... */
00127
      fclose(out);
00128
       /* Free... */
00129
00130
      free (met);
00131
00132
      return EXIT_SUCCESS;
00133 }
```

Here is the call graph for this function:



## 5.20 met\_map.c

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

5.20 met map.c 149

```
MPTRAC is free software: you can redistribute it and/or modify
00005
         it under the terms of the GNU General Public License as published by
00006
         the Free Software Foundation, either version 3 of the License, or
00007
         (at your option) any later version.
00008
00009
        MPTRAC is distributed in the hope that it will be useful,
        but WITHOUT ANY WARRANTY; without even the implied warranty of
00010
00011
        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
        GNU General Public License for more details.
00013
        You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
00016
        Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00017
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028
        int argc,
00029
        char *argv[]) {
00030
00031
        ctl t ctl;
00032
00033
        met_t *met;
00034
00035
        FILE *out;
00036
        static double dz, dzmin = 1e10, z, timem[EX][EY], psm[EX][EY], tm[EX][EY], um[EX][EY], vm[EX][EY], wm[EX][EY], h2om[EX][EY], o3m[EX][EY];
00037
00038
00039
00040
        static int i, ip, ip2, ix, iy, np[EX][EY];
00041
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);
00050
        z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00051
00052
00053
         /* Loop over files... *
00054
         for (i = 3; i < argc; i++) {</pre>
00055
00056
           /\star Read meteorological data... \star/
00057
           read_met(&ctl, argv[i], met);
00058
00059
           /* Find nearest pressure level... */
00060
           for (ip2 = 0; ip2 < met->np; ip2++) {
00061
             dz = fabs(Z(met->p[ip2]) - z);
             if (dz < dzmin) {
  dzmin = dz;</pre>
00062
00063
00064
               ip = ip2;
00065
00066
00067
00068
           /* Average data... */
           for (ix = 0; ix < met->nx; ix++)
  for (iy = 0; iy < met->ny; iy++) {
    timem[ix][iy] += met->time;
00069
00070
00071
00072
               tm[ix][iy] += met->t[ix][iy][ip];
               um[ix][iy] += met->u[ix][iy][ip];
00073
00074
               vm[ix][iy] += met->v[ix][iy][ip];
00075
               wm[ix][iy] += met->w[ix][iy][ip];
00076
               h2om[ix][iy] += met->h2o[ix][iy][ip];
               o3m[ix][iy] += met->o3[ix][iy][ip];
psm[ix][iy] += met->ps[ix][iy];
00077
00078
00079
               np[ix][iy]++;
00080
00081
        }
00082
00083
        /* Create output file... */
        printf("Write meteorological data file: %s\n", argv[2]);
00084
00085
         if (!(out = fopen(argv[2], "w")))
00086
          ERRMSG("Cannot create file!");
00087
00088
         /* Write header... */
00089
        fprintf(out,
                  "# $1
00090
                         = time [s]\n"
00091
                  "# $2
                         = altitude [km]\n"
00092
                  "# $3
                         = longitude [deg]\n"
                  "# $4 = latitude [deg]\n"
00093
                  "# $5 = pressure [hPa] \n"
00094
00095
                  "# $6 = temperature [K]\n"
```

```
"# $7 = zonal wind [m/s]\n"
00097
             "# $8 = meridional wind [m/s]\n"
             "# $9 = vertical wind [hPa/s]\n"
00098
             "# $10 = H2O volume mixing ratio [1]\n"
00099
             "# $11 = 03 volume mixing ratio [1]\n'
"# $12 = surface pressure [hPa]\n");
00100
00101
00102
00103
      00104
00105
00106
00107
00108
00109
00110
00111
00112
00113
00114
00115
        for (ix = 0; ix < met->nx; ix++)
         00116
00117
00118
00119
00120
00121
00122
00123
                  psm[ix][iy] / np[ix][iy]);
00124
00125
00126
      /* Close file... */
00127
      fclose(out);
00128
00129
      /* Free... */
00130
      free (met);
00131
00132
      return EXIT SUCCESS;
00133 }
```

## 5.21 met prof.c File Reference

Extract vertical profile from meteorological data.

**Functions** 

• int main (int argc, char \*argv[])

### 5.21.1 Detailed Description

Extract vertical profile from meteorological data.

Definition in file met\_prof.c.

# 5.21.2 Function Documentation

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

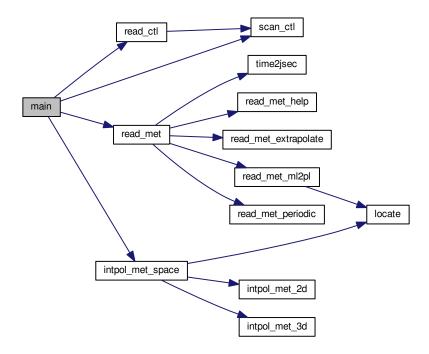
Definition at line 38 of file met\_prof.c.

```
00040
00041
00042
           ctl_t ctl;
00043
00044
           met t *met;
00045
           FILE *out;
00047
           static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ],
w, wm[NZ], h2o, h2om[NZ], o3, o3m[NZ], ps, psm[NZ];
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... */
              read_met(&ctl, argv[i], met);
00078
00079
08000
               for (z = z0; z \le z1; z += dz) {
                 iz = (int) ((z - z0) / dz);

if (iz < 0 || iz > NZ)
00081
00082
00083
                     ERRMSG("Too many altitudes!");
                  for (lon = lon0; lon <= lon1; lon += dlon)</pre>
00085
                     for (lat = lat0; lat <= lat1; lat += dlat) {</pre>
00086
                       intpol_met_space(met, P(z), lon, lat, &ps,
                        00087
00088
00089
00090
                           timem[iz] += met->time;
00091
                           lonm[iz] += lon;
                           latm[iz] += lat;
00092
                           tm[iz] += t;
um[iz] += u;
00093
00094
00095
                           vm[iz] += v;
                           wm[iz] += w;
00097
                           h2om[iz] += h2o;
00098
                           o3m[iz] += o3;
00099
                           psm[iz] += ps;
00100
                           np[iz]++;
00101
00102
                     }
00103
              }
00104
00105
00106
            /* Normalize... */
           for (z = z0; z <= z1; z += dz) {
  iz = (int) ((z - z0) / dz);
00107
00108
              if (np[iz] > 0) {
00109
00110
                 timem[iz] /= np[iz];
                 lonm[iz] /= np[iz];
latm[iz] /= np[iz];
00111
00112
                 tm[iz] /= np[iz];
um[iz] /= np[iz];
vm[iz] /= np[iz];
00113
00114
00115
00116
                  wm[iz] /= np[iz];
00117
                 h2om[iz] /= np[iz];
                 o3m[iz] /= np[iz];
psm[iz] /= np[iz];
00118
00119
00120
              } else {
                 timem[iz] = GSL_NAN;
                  lonm[iz] = GSL_NAN;
latm[iz] = GSL_NAN;
00122
00123
                 tm[iz] = GSL_NAN;
um[iz] = GSL_NAN;
00124
00125
                  vm[iz] = GSL_NAN;
00126
```

```
00127
               wm[iz] = GSL_NAN;
               h2om[iz] = GSL_NAN;
o3m[iz] = GSL_NAN;
psm[iz] = GSL_NAN;
00128
00129
00130
00131
00132
00133
00134
           /* Create output file... */
00135
          printf("Write meteorological data file: sn'', argv[2]);
          if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");
00136
00137
00138
00139
          /* Write header... */
00140
          fprintf(out,
00141
                    "# $1
                             = time [s]\n"
                    "# $1 = time [s]\n"
"# $2 = altitude [km]\n"
"# $3 = longitude [deg]\n"
"# $4 = latitude [deg]\n"
"# $5 = pressure [hPa]\n"
00142
00143
00144
00145
00146
                    "# $6 = temperature [K]\n"
00147
                     "# $7 = zonal wind [m/s] \n"
                     "# $8 = meridional wind [m/s]\n"
00148
                     "# $9 = vertical wind [hPa/s]\n"
00149
                    "# $10 = H2O volume mixing ratio [1]\n"
"# $11 = O3 volume mixing ratio [1]\n"
"# $12 = surface pressure [hPa]\n\n");
00150
00151
00152
00153
          /* Write data... */
for (z = z0; z <= z1; z += dz) {
  iz = (int) ((z - z0) / dz);
00154
00155
00156
            00157
00158
00159
00160
00161
          /* Close file... */
00162
          fclose(out);
00163
00164
00165
           /* Free... */
00166
          free(met);
00167
          return EXIT_SUCCESS;
00168
00169 }
```

Here is the call graph for this function:



5.22 met prof.c 153

# 5.22 met\_prof.c

```
00001 /*
          This file is part of MPTRAC.
00003
00004
           MPTRAC is free software: you can redistribute it and/or modify
00005
           it under the terms of the GNU General Public License as published by
           the Free Software Foundation, either version 3 of the License, or
00006
00007
           (at your option) any later version.
80000
00009
           MPTRAC is distributed in the hope that it will be useful,
00010
           but WITHOUT ANY WARRANTY; without even the implied warranty of
00011
           MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
           GNU General Public License for more details.
00013
          You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
00016
00017
           Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00027 /*
           Dimensions...
00028
00029
00030
00031 /* Maximum number of altitudes. */
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 *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>   <met0> [ <met1> ... ]");
00060
00061
          /* 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, "L0N0", -1, "0", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "L0N1", -1, "0", NULL);
dlon = scan_ctl(argv[1], argc, argv, "DL0N", -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);
           /* Read control parameters... */
00062
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
             read_met(&ctl, argv[i], met);
00078
00079
              /* Average... */
              for (z = z0; z <= z1; z += dz) {
  iz = (int) ((z - z0) / dz);
00080
00081
                 if (iz < 0 || iz > NZ)
00082
00083
                   ERRMSG("Too many altitudes!");
00084
                 for (lon = lon0; lon <= lon1; lon += dlon)</pre>
                  for (lat = lat0; lat <= lat1; lat += dlat) {
00085
00086
                     00087
00088
00089
                           && gsl_finite(v) && gsl_finite(w)) {
```

```
timem[iz] += met->time;
                     lonm[iz] += lon;
latm[iz] += lat;
00091
00092
                     tm[iz] += t;
00093
                     um[iz] += u;
00094
00095
                     vm[iz] += v;
                     wm[iz] += w;
00097
                     h2om[iz] += h2o;
                     o3m[iz] += o3;
psm[iz] += ps;
00098
00099
00100
                     np[iz]++;
00101
                   }
00102
                }
00103
           }
00104
00105
00106
         /* Normalize... */
         for (z = z0; z <= z1; z += dz) {
  iz = (int) ((z - z0) / dz);
00107
00108
           if (np[iz] > 0) {
00109
             timem[iz] /= np[iz];
timem[iz] /= np[iz];
lonm[iz] /= np[iz];
latm[iz] /= np[iz];
tm[iz] /= np[iz];
um[iz] /= np[iz];
00110
00111
00112
00113
00114
00115
              vm[iz] /= np[iz];
              wm[iz] /= np[iz];
00116
             h2om[iz] /= np[iz];
o3m[iz] /= np[iz];
psm[iz] /= np[iz];
00117
00118
00119
00120
           } else {
00121
              timem[iz] = GSL_NAN;
             lonm[iz] = GSL_NAN;
latm[iz] = GSL_NAN;
00122
00123
             tm[iz] = GSL_NAN;
um[iz] = GSL_NAN;
00124
00125
             vm[iz] = GSL_NAN;
vm[iz] = GSL_NAN;
00126
00128
              h2om[iz] = GSL_NAN;
00129
              o3m[iz] = GSL_NAN;
00130
              psm[iz] = GSL_NAN;
00131
00132
00133
00134
         /* Create output file... */
00135
         printf("Write meteorological data file: %s\n", argv[2]);
00136
         if (!(out = fopen(argv[2], "w")))
00137
           ERRMSG("Cannot create file!");
00138
00139
         /* Write header... */
00140
         fprintf(out,
00141
                   "# $1 = time [s] \n"
                   "# $2 = altitude [km]\n"
00142
                  "# $3 = longitude [deg]\n"
"# $4 = latitude [deg]\n"
"# $5 = pressure [hPa]\n"
"# $6 = temperature [K]\n"
00143
00144
00145
00147
                   "# $7 = zonal wind [m/s]\n"
00148
                   "# $8 = meridional wind [m/s]\n"
                   "# $9 = vertical wind [hPa/s]\n"
00149
                   "# $10 = H2O volume mixing ratio [1]\n"
"# $11 = O3 volume mixing ratio [1]\n"
00150
00151
00152
                   "# $12 = surface pressure [hPa]\n\n");
00153
         /* Write data... */
00154
        00155
00156
00157
00158
00159
00160
00161
00162
         /* Close file... */
00163
         fclose(out);
00164
00166
         free (met);
00167
00168
         return EXIT_SUCCESS;
00169 }
```

# 5.23 met\_sample.c File Reference

Sample meteorological data at given geolocations.

#### **Functions**

• int main (int argc, char \*argv[])

### 5.23.1 Detailed Description

Sample meteorological data at given geolocations.

Definition in file met\_sample.c.

#### 5.23.2 Function Documentation

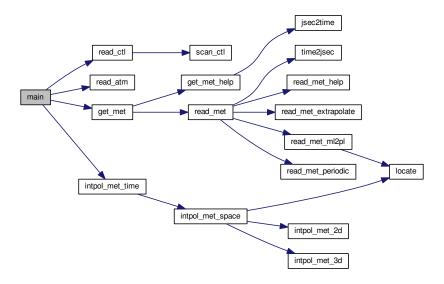
```
5.23.2.1 int main ( int argc, char * argv[] )
```

Definition at line 31 of file met sample.c.

```
00033
00034
00035
        ctl_t ctl;
00036
00037
        atm t *atm;
00038
00039
        met_t *met0, *met1;
00040
00041
        FILE *out;
00042
00043
        double t, u, v, w, h2o, o3;
00044
00045
        int ip;
00046
00047
        /* Check arguments... */
00048
        if (argc < 4)
00049
          ERRMSG("Give parameters: <ctl> <metbase> <atm_in> <sample.tab>");
00050
00051
        /* Allocate... */
00052
        ALLOC(atm, atm_t, 1);
00053
        ALLOC(met0, met_t, 1);
00054
        ALLOC(met1, met_t, 1);
00055
00056
        /* Read control parameters... */
00057
        read_ctl(argv[1], argc, argv, &ctl);
00058
00059
        /\star Read atmospheric data... \star/
00060
        read_atm(argv[3], &ctl, atm);
00061
00062
        /* Create output file... */
00063
        printf("Write meteorological data file: %s\n", argv[4]);
00064
        if (!(out = fopen(argv[4], "w")))
00065
          ERRMSG("Cannot create file!");
00066
        /* Write header... */
00067
00068
        fprintf(out,
                 "# $1 = time [s] \n"
00069
00070
                 "# $2 = altitude [km] \n"
00071
                 "# $3 = longitude [deg] \n"
                 "# $4 = latitude [deg]\n"
00072
                 "# $5 = pressure [hPa]\n"
00073
                 "# $6 = temperature [K]\n"
"# $7 = zonal wind [m/s]\n"
00074
00075
00076
                 "# $8 = meridional wind [m/s]\n"
00077
                 "# $9 = vertical wind [hPa/s]\n"
                 "# $10 = H20 volume mixing ratio [1]\n" "# $11 = 03 volume mixing ratio [1]\n\n");
00078
00079
00080
        /* Loop over air parcels... */
for (ip = 0; ip < atm->np; ip++) {
00081
00082
00083
00084
           /* Get meteorological data... */
00085
          get_met(&ctl, argv[2], atm->time[ip], met0, met1);
00086
00087
           /* Interpolate meteorological data... */
           intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
00088
      lon[ip],
```

```
00089
                      atm->lat[ip], NULL, &t, &u, &v, &w, &h2o, &o3);
00090
        00091
00092
00093
00094
00095
00096
00097
      /\star Close file... \star/
00098
      fclose(out);
00099
00100
      /* Free... */
00101
      free(atm);
00102
      free (met0);
00103
      free (met1);
00104
      return EXIT_SUCCESS;
00105
00106 }
```

Here is the call graph for this function:



## 5.24 met\_sample.c

```
00001 /*
          This file is part of MPTRAC.
00002
00003
          MPTRAC is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by
00004
00005
00006
          the Free Software Foundation, either version 3 of the License, or
00007
          (at your option) any later version.
00008
          MPTRAC is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
00009
00010
          MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00011
00012
          GNU General Public License for more details.
00013
          You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
00016
00017
          Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /*
00028
           Main...
00029
00030
```

```
00031 int main(
00032
       int argc,
00033
       char *argv[]) {
00034
00035
       ctl_t ctl;
00036
       atm_t *atm;
00038
00039
       met_t *met0, *met1;
00040
00041
       FILE *out;
00042
00043
       double t, u, v, w, h2o, o3;
00044
00045
00046
00047
       /* Check arguments... */
00048
       if (argc < 4)
         ERRMSG("Give parameters: <ctl> <metbase> <atm_in> <sample.tab>");
00049
00050
00051
00052
       ALLOC(atm, atm_t, 1);
00053
       ALLOC(met0, met_t, 1);
00054
       ALLOC(met1, met_t, 1);
00055
00056
       /* Read control parameters... */
00057
       read_ctl(argv[1], argc, argv, &ctl);
00058
00059
       /* Read atmospheric data... */
00060
       read_atm(argv[3], &ctl, atm);
00061
00062
        /* Create output file... */
00063
       printf("Write meteorological data file: %s\n", argv[4]);
00064
       if (!(out = fopen(argv[4], "w")))
00065
         ERRMSG("Cannot create file!");
00066
       /* Write header... */
00067
00068
       fprintf(out,
00069
               "# $1
                      = time [s]\n"
00070
               "# $2 = altitude [km]\n"
               "# $3 = longitude [deg] \n"
00071
               "# $4 = latitude [deg]\n"
"# $5 = pressure [hPa]\n"
00072
00073
               "# $6 = temperature [K]\n"
00074
00075
               "# $7 = zonal wind [m/s]\n"
00076
                "# $8 = meridional wind [m/s]\n"
00077
                "# $9 = vertical wind [hPa/s] n"
                "# $10 = H2O volume mixing ratio [1]\n"
00078
                "# $11 = 03 volume mixing ratio [1]\n\n");
00079
00080
00081
       /* Loop over air parcels... */
00082
       for (ip = 0; ip < atm->np; ip++) {
00083
00084
          /\star Get meteorological data... \star/
         get_met(&ctl, argv[2], atm->time[ip], met0, met1);
00085
00086
         /* Interpolate meteorological data... */
00088
         intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
     lon[ip],
00089
                          atm->lat[ip], NULL, &t, &u, &v, &w, &h2o, &o3);
00090
         00091
00092
00093
                 atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
00094
                 atm->p[ip], t, u, v, w, h2o, o3);
00095
00096
00097
       /* Close file... */
00098
       fclose(out);
00100
       /* Free... */
00101
       free (atm);
00102
       free (met0);
00103
       free (met1);
00104
00105
       return EXIT_SUCCESS;
00106 }
```

## 5.25 met zm.c File Reference

Extract zonal mean from meteorological data.

#### **Functions**

• int main (int argc, char \*argv[])

### 5.25.1 Detailed Description

Extract zonal mean from meteorological data.

Definition in file met zm.c.

#### 5.25.2 Function Documentation

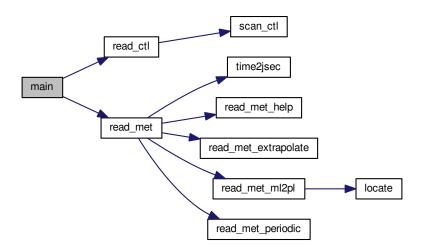
```
5.25.2.1 int main ( int argc, char * argv[])
```

Definition at line 27 of file met zm.c.

```
00029
00030
00031
          ctl_t ctl;
00032
00033
         met t *met;
00034
00035
          FILE *out;
00036
00037
          static double timem[EP][EY], psm[EP][EY], 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... */
         if (argc < 4)
00048
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... */
00058
            read_met(&ctl, argv[i], met);
00059
00060
             /* Average data... */
00061
            for (ix = 0; ix < met->nx; ix++)
               for (iy = 0; iy < met->ny; iy++)
00062
                 for (ip = 0; ip < met->np; ip++) {
  timem[ip][iy] += met->time;
00063
00064
                    tm[ip][iy] += met ->t[ix][iy][ip];
um[ip][iy] += met ->u[ix][iy][ip];
00065
00066
00067
                    vm[ip][iy] += met->v[ix][iy][ip];
00068
                    + gsl_pow_2(met->v[ix][iy][ip]));
wm[ip][iy] += met->w[ix][iy][ip];
h2om[ip][iy] += met->h2o[ix][iy][ip];
00069
00070
00071
                    o3m[ip][iy] += met->o3[ix][iy][ip];
00072
00073
                    psm[ip][iy] += met->ps[ix][iy];
00074
                    \label{eq:tm2[ip][iy] += gsl_pow_2(met->t[ix][iy][ip]);} tm2[ip][iy] += gsl_pow_2(met->t[ix][iy][ip]);
00075
                    \label{limiting} \verb"um2[ip][iy] += gsl_pow_2 (met->u[ix][iy][ip]);
                    vm2[ip][iy] += gsl_pow_2(met->v[ix][iy][ip]);
vm2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip])
+ gsl_pow_2(met->v[ix][iy][ip]);
00076
00077
00078
00079
                    wm2[ip][iy] += gsl_pow_2(met->w[ix][iy][ip]);
00080
                    h2om2[ip][iy] += gsl_pow_2(met->h2o[ix][iy][ip]);
                    o3m2[ip][iy] += gsl_pow_2(met->o3[ix][iy][ip]);
psm2[ip][iy] += gsl_pow_2(met->ps[ix][iy]);
00081
00082
00083
                    np[ip][iy]++;
00084
00085
          }
```

```
00086
00087
         /* Create output file... */
00088
         printf("Write meteorological data file: %s\n", argv[2]);
         if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");
00089
00090
00091
00092
         /* Write header... */
00093
         fprintf(out,
                          = time [s] n
00094
                   "# $1
                   "# $2 = altitude [km]\n"
"# $3 = latitude [deg]\n"
00095
00096
                   "# $4 = temperature mean [K]\n"
00097
00098
                   "# $5 = temperature standard deviation [K]\n"
00099
                   "# $6 = zonal wind mean [m/s]\n"
00100
                   "# $7 = zonal wind standard deviation [m/s]\n"
                  "# $8 = meridional wind mean [m/s]\n"  
"# $9 = meridional wind standard deviation [m/s]\n"
00101
00102
                   "# $10 = horizontal wind mean [m/s]\n"
00103
                   "# $11 = horizontal wind standard deviation [m/s]\n"
00104
                  "# $12 = vertical wind mean [hPa/s]\n"
00105
00106
                   "# $13 = vertical wind standard deviation [hPa/s]\n"
                   "# $14 = H20 \text{ vmr mean } [1] \n"
00107
                   "# $15 = H20 \text{ vmr standard deviation } [1] \n"
00108
                   "# $16 = O3 vmr mean [1]\n"
00109
                   "# $17 = 03 vmr standard deviation [1]\n"
00110
                   "# $18 = surface pressure mean [hPa]\n"
00111
00112
                   "# $19 = surface pressure standard deviation [hPa] \n");
00113
00114
         /* Write data... */
        for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
00115
00116
           00117
00118
00119
                        timem[ip][iy] / np[ip][iy], Z(met->p[ip]), met->lat[iy],
tm[ip][iy] / np[ip][iy],
sqrt(tm2[ip][iy] / np[ip][iy] -
    gsl_pow_2(tm[ip][iy] / np[ip][iy])),
00120
00121
00122
00124
                       um[ip][iy] / np[ip][iy],
00125
                       sqrt(um2[ip][iy] / np[ip][iy] -
00126
                             gsl_pow_2(um[ip][iy] / np[ip][iy])),
                       vm[ip][iy] / np[ip][iy],
sqrt(vm2[ip][iy] / np[ip][iy] -
00127
00128
                       sqt(vm2[ip][iy] / np[ip][iy] /
    gs1_pow_2(vm[ip][iy] / np[ip][iy])),
vhm[ip][iy] / np[ip][iy],
sqrt(vhm2[ip][iy] / np[ip][iy] -
00129
00130
00131
                       00132
00133
00134
00135
00136
00137
00138
                             gsl_pow_2(h2om[ip][iy] / np[ip][iy])),
                       o3m[ip][iy] / np[ip][iy],
sqrt(o3m2[ip][iy] / np[ip][iy] -
00139
00140
                       gsl_pow_2(o3m[ip][iy] / np[ip][iy])),
psm[ip][iy] / np[ip][iy],
sqrt(psm2[ip][iy] / np[ip][iy] -
00141
00143
00144
                             gsl_pow_2(psm[ip][iy] / np[ip][iy]));
00145
00146
00147
        /* Close file... */
00148
        fclose(out);
00149
00150
         /* Free... */
00151
        free(met);
00152
         return EXIT SUCCESS:
00153
00154 }
```

Here is the call graph for this function:



# 5.26 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 *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);
```

5.26 met zm.c 161

```
00054
         /* Loop over files... */
00055
        for (i = 3; i < argc; i++) {</pre>
00056
00057
           /* Read meteorological data... */
00058
           read met (&ctl, argv[i], met);
00060
00061
           for (ix = 0; ix < met->nx; ix++)
00062
             for (iy = 0; iy < met->ny; iy++)
               for (ip = 0; ip < met->np; ip++) {
00063
                 timem[ip][iy] += met->time;
00064
00065
                 tm[ip][iy] += met->t[ix][iy][ip];
                 um[ip][iy] += met->u[ix][iy][ip];
00066
00067
                 vm[ip][iy] += met->v[ix][iy][ip];
00068
                 00069
                                        + gsl_pow_2 (met->v[ix][iy][ip]));
                 wm[ip][iy] += met->w[ix][iy][ip];
h2om[ip][iy] += met->h2o[ix][iy][ip];
00070
                 o3m[ip][iy] += met->o3[ix][iy][ip];
00072
00073
                 psm[ip][iy] += met->ps[ix][iy];
00074
                  tm2[ip][iy] += gsl_pow_2(met->t[ix][iy][ip]);
00075
                 \label{limit} $$ um2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip]);$
                 vm2[ip][iy] += gsl_pow_2(met->v[ix][iy][ip]);
vhm2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip])
00076
00077
                    + gsl_pow_2 (met->v[ix][iy][ip]);
00078
00079
                  wm2[ip][iy] += gsl_pow_2(met->w[ix][iy][ip]);
00080
                 \label{eq:h2om2[ip][iy] += gsl_pow_2(met->h2o[ix][iy][ip]);} h2om2[ip][iy] += gsl_pow_2(met->h2o[ix][iy][ip]);}
                 o3m2[ip][iy] += gsl_pow_2(met->o3[ix][iy][ip]);
psm2[ip][iy] += gsl_pow_2(met->ps[ix][iy]);
00081
00082
00083
                 np[ip][iy]++;
00084
00085
00086
        /* Create output file... */
printf("Write meteorological data file: %s\n", argv[2]);
if (!(out = fopen(argv[2], "w")))
00087
00088
00089
           ERRMSG("Cannot create file!");
00091
00092
         /* Write header... */
        00093
00094
                 "# $2 = altitude [km]\n"
"# $3 = latitude [deg]\n"
                  "# $2
00095
00096
                        = temperature mean [K]\n"
00097
00098
                  "# $5 = temperature standard deviation [K]\n"
00099
                  "# $6 = zonal wind mean [m/s]\n"
                  "# \$7 = zonal wind standard deviation [m/s]\n"
00100
                 "# $8 = meridional wind mean [m/s]\n"
00101
                        = meridional wind standard deviation [m/s]\n"
                 "# $9
00102
00103
                 "# $10 = horizontal wind mean [m/s]\n"
                 "# $11 = horizontal wind standard deviation [m/s]\n"
00104
00105
                 "# $12 = vertical wind mean [hPa/s]\n"
00106
                 "# $13 = vertical wind standard deviation [hPa/s]\n"
                  "# $14 = H20 vmr mean [1]\n"
00107
                 "# $15 = H20 vmr standard deviation [1]\n"
00108
                 "# $16 = 03 \text{ vmr mean } [1] \n"
00110
                 "# $17 = 03 vmr standard deviation [1]\n"
00111
                  "# $18 = surface pressure mean [hPa] \n"
00112
                  "# $19 = surface pressure standard deviation [hPa] \n");
00113
        /* Write data... */
00114
        for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
00115
00116
           00117
00118
                      00119
00120
00121
00123
00124
                      um[ip][iy] / np[ip][iy],
                      00125
00126
00127
00128
00129
                            gsl_pow_2(vm[ip][iy] / np[ip][iy])),
                      ysl_pow_z(vm[zp[iy], np[ip][iy],

vhm[ip][iy] / np[ip][iy],

sqrt(vhm2[ip][iy] / np[ip][iy] -

gsl_pow_2(vhm[ip][iy] / np[ip][iy]),

wm[ip][iy] / np[ip][iy],

cont(vm2[in][in] / np[in][iv]
00130
00131
00132
00133
                      wm[tp](ty) / np[tp](iy] -
gsl_pow_2(wm[ip][iy] / np[ip][iy])),
h2om[ip][iy] / np[ip][iy],
00135
00136
                      00137
00138
00139
```

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

# 5.27 smago.c File Reference

Estimate horizontal diffusivity based on Smagorinsky theory.

## **Functions**

• int main (int argc, char \*argv[])

### 5.27.1 Detailed Description

Estimate horizontal diffusivity based on Smagorinsky theory.

Definition in file smago.c.

# 5.27.2 Function Documentation

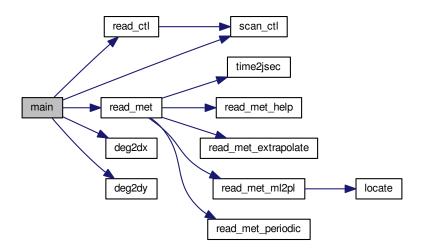
## 5.27.2.1 int main ( int *argc*, char \* *argv*[])

Definition at line 8 of file smago.c.

```
00010
                       {
00011
00012
        ctl_t ctl;
00013
00014
        met_t *met;
00015
00016
        FILE *out;
00017
00018
        static double dz, dzmin = 1e10, z, t, s, ls2, k[EX][EY], c = 0.15;
00019
00020
        static int ip, ip2, ix, iy;
00021
00022
00023
        /* Allocate... */
        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);
00029
00030
00031
        z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00032
00033
        /* Read meteorological data... */
00034
        read_met(&ctl, argv[3], met);
00035
00036
        /\star Find nearest pressure level... \star/
00037
        for (ip2 = 0; ip2 < met->np; ip2++) {
00038
          dz = fabs(Z(met->p[ip2]) - z);
```

```
00039
          if (dz < dzmin) {</pre>
00040
           dzmin = dz;
00041
            ip = ip2;
00042
00043
00044
        /* Write info... */
00046
        printf("Analyze %g hPa...\n", met->p[ip]);
00047
00048
        /\star Calculate horizontal diffusion coefficients... \star/
        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])));
            s = 0.5 * ((met->u[ix][iy + 1][ip] - met->u[ix][iy - 1][ip])
00056
                        / (1000. * deg2dy(met->lat[iy + 1] - met->lat[iy - 1]))
00057
00058
                        + (met->v[ix + 1][iy][ip] - met->v[ix - 1][iy][ip])
00059
                        / (1000. *
00060
                           deg2dx(met->lon[ix + 1] - met->lon[ix - 1],
00061
                                  met->lat[iy])));
00062
            ls2 = gsl_pow_2(c * 500. * deg2dy(met->lat[iy + 1] - met->lat[iy - 1]));
            if (fabs(met->lat[iy]) > 80)
00063
00064
              1s2 \star = (90. - fabs(met->lat[iy])) / 10.;
00065
            k[ix][iy] = 1s2 * sqrt(2.0 * (gsl_pow_2(t) + gsl_pow_2(s)));
00066
00067
        /* Create output file... */
printf("Write data file: %s\n", argv[2]);
00068
00069
00070
        if (!(out = fopen(argv[2], "w")))
00071
          ERRMSG("Cannot create file!");
00072
00073
        /* Write header... */
00074
        fprintf(out,
                 "# $1 = longitude [deg]\n"
00076
                 "# $2 = latitude [deg] n"
00077
                 "# $3 = zonal wind [m/s] \n"
                 "# $4 = meridional wind [m/s]\n"
00078
                 "# $5 = horizontal diffusivity [m^2/s]\n");
00079
00080
00081
        /* Write data... */
        for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
  for (ix = 0; ix < met->nx; ix++)
00082
00083
00084
            00085
00086
00087
                       met->u[ix][iy][ip], met->v[ix][iy][ip], k[ix][iy]);
00088
00089
           for (ix = 0; ix < met->nx; ix++)
            00090
00091
00092
00093
                       met->u[ix][iy][ip], met->v[ix][iy][ip], k[ix][iy]);
00094
00095
00096
        /* Close file... */
00097
        fclose(out);
00098
00099
        /* Free... */
00100
        free (met);
00101
00102
        return EXIT_SUCCESS;
00103 }
```

Here is the call graph for this function:



# 5.28 smago.c

```
00001
00006 #include "libtrac.h"
00007
00008 int main(
00009
        int argc,
         char *argv[]) {
00010
00011
00012
         ctl t ctl:
00013
00014
         met_t *met;
00015
00016
         FILE *out;
00017
00018
         static double dz, dzmin = 1e10, z, t, s, 1s2, k[EX][EY], c = 0.15;
00019
00020
         static int ip, ip2, ix, iy;
00021
00022
         /* Allocate... */
00023
         ALLOC(met, met_t, 1);
00024
00025
         /* Check arguments... */
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)
ls2 *= (90. - fabs(met->lat[iy])) / 10.;
00063
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
        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
00081
        /* Write data... */
for (iy = 0; iy < met->ny; iy++) {
  fprintf(out, "\n");
00082
00083
00084
           for (ix = 0; ix < met->nx; ix++)
            00085
00086
00087
                        met->u[ix][iy][ip], met->v[ix][iy][ip], k[ix][iy]);
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
                        \label{eq:met-v} \text{met->} u \texttt{[ix][iy][ip], met->} v \texttt{[ix][iy][ip], k[ix][iy]);}
00093
00094
00095
00096
         /* Close file... */
00097
        fclose(out);
00098
00099
        /* Free... */
00100
        free (met);
00101
00102
        return EXIT_SUCCESS;
00103 }
```

# 5.29 split.c File Reference

Split air parcels into a larger number of parcels.

### **Functions**

• int main (int argc, char \*argv[])

### 5.29.1 Detailed Description

Split air parcels into a larger number of parcels.

Definition in file split.c.

#### 5.29.2 Function Documentation

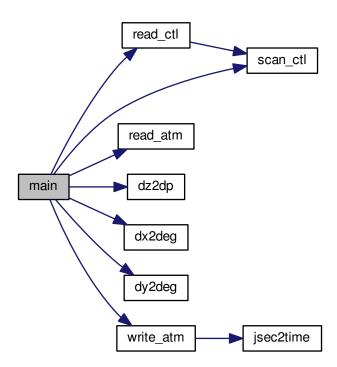
### 5.29.2.1 int main ( int argc, char \* argv[] )

Definition at line 27 of file split.c.

```
00029
00030
00031
          atm_t *atm, *atm2;
00032
00033
          ctl t ctl;
00034
          gsl_rng *rng;
00036
00037
          double m, mtot = 0, dt, dx, dz, mmax = 0,
00038
            t0, t1, z0, z1, lon0, lon1, lat0, lat1;
00039
00040
          int i, ip, iq, n;
00041
00042
          /* Allocate... */
00043
          ALLOC(atm, atm_t, 1);
00044
          ALLOC(atm2, atm_t, 1);
00045
00046
          /* Check arguments... */
00047
              (argc < 4)
00048
            ERRMSG("Give parameters: <ctl> <atm_in> <atm_out>");
00049
00050
          /* Read control parameters... */
         read_ctl(argv[1], argc, argv, &ctl);
n = (int) scan_ctl(argv[1], argc, argv, "SPLIT_N", -1, "", NULL);
m = scan_ctl(argv[1], argc, argv, "SPLIT_M", -1, "-999", NULL);
dt = scan_ctl(argv[1], argc, argv, "SPLIT_DT", -1, "0", NULL);
00051
00052
00053
          to = scan_ctl(argv[1], argc, argv, "SPLIT_TO", -1, "0", NULL);
t1 = scan_ctl(argv[1], argc, argv, "SPLIT_T1", -1, "0", NULL);
00055
00056
          dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);
z0 = scan_ctl(argv[1], argc, argv, "SPLIT_Z0", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "SPLIT_Z1", -1, "0", NULL);
00057
00058
00059
          dx = scan_ctl(argv[1], argc, argv, "SPLIT_DX", -1, "0", NULL);
00060
         lon0 = scan_ctl(argv[1], argc, argv, "SPLIT_LON0", -1, "0", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "SPLIT_LON1", -1, "0", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT0", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT1", -1, "0", NULL);
00061
00062
00063
00064
00065
00066
          /* Init random number generator... */
          gsl_rng_env_setup();
00067
00068
          rng = gsl_rng_alloc(gsl_rng_default);
00069
00070
          /* Read atmospheric data... */
00071
          read_atm(argv[2], &ctl, atm);
00072
00073
          /* Get total and maximum mass... */
00074
          if (ctl.qnt_m >= 0)
00075
            for (ip = 0; ip < atm->np; ip++) {
               mtot += atm->q[ctl.qnt_m][ip];
00076
00077
               mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
00078
00079
          if (m > 0)
00080
            mtot = m;
00081
00082
          /* Loop over air parcels... */
00083
          for (i = 0; i < n; i++) {
00084
00085
             /* Select air parcel... */
00086
             if (ctl.qnt_m >= 0)
00087
               do {
00088
                 ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00089
               } while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
00090
             else
00091
               ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00092
00093
             /* Set time... */
00094
             if (t1 > t0)
00095
               atm2 - time[atm2 - np] = t0 + (t1 - t0) * qsl_rnq_uniform_pos(rnq);
00096
             else
00097
              atm2->time[atm2->np] = atm->time[ip]
00098
                 + gsl_ran_gaussian_ziggurat(rng, dt / 2.3548);
00099
00100
             /* Set vertical position... */
             if (z1 > z0)
00101
00102
               atm2 \rightarrow p[atm2 \rightarrow np] = P(z0 + (z1 - z0) * gsl_rng_uniform_pos(rng));
00103
00104
               atm2->p[atm2->np] = atm->p[ip]
```

```
+ dz2dp(gsl_ran_gaussian_ziggurat(rng, dz / 2.3548), atm->p[ip]);
00106
00107
            /\star Set horizontal position... \star/
           if (lon1 > lon0 && lat1 > lat0) {
00108
             atm2->lon(atm2->np) = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
atm2->lat(atm2->np) = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
00109
00110
00111
00112
            atm2 \rightarrow lon[atm2 \rightarrow np] = atm \rightarrow lon[ip]
00113
                + gsl_ran_gaussian_ziggurat(rng, dx2deg(dx, atm->lat[ip]) / 2.3548);
00114
             atm2->lat[atm2->np] = atm->lat[ip]
                + gsl_ran_gaussian_ziggurat(rng, dy2deg(dx) / 2.3548);
00115
00116
00117
00118
            /* Copy quantities... */
00119
           for (iq = 0; iq < ctl.nq; iq++)
            atm2->q[iq][atm2->np] = atm->q[iq][ip];
00120
00121
           /* Adjust mass... */
if (ctl.qnt_m >= 0)
00122
00123
00124
             atm2->q[ct1.qnt_m][atm2->np] = mtot / n;
00125
           /* Increment particle counter... */
if ((++atm2->np) >= NP)
00126
00127
             ERRMSG("Too many air parcels!");
00128
00129
00130
00131
         /\star Save data and close file... \star/
00132
         write_atm(argv[3], &ctl, atm2, atm->time[0]);
00133
         /* Free... */
00134
00135
         free (atm);
00136
         free(atm2);
00137
00138
         return EXIT_SUCCESS;
00139 }
```

Here is the call graph for this function:



## 5.30 split.c

```
00001 /*
          This file is part of MPTRAC.
00003
00004
           MPTRAC is free software: you can redistribute it and/or modify
00005
           it under the terms of the GNU General Public License as published by
           the Free Software Foundation, either version 3 of the License, or
00006
00007
           (at your option) any later version.
80000
00009
           MPTRAC is distributed in the hope that it will be useful,
00010
           but WITHOUT ANY WARRANTY; without even the implied warranty of
00011
           MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
           GNU General Public License for more details.
00013
          You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
00016
00017
           Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028 int argc,
00029
          char *argv[]) {
00030
00031
          atm t *atm, *atm2;
00032
00033
          ctl_t ctl;
00034
00035
          gsl_rng *rng;
00036
          double m, mtot = 0, dt, dx, dz, mmax = 0,
   t0, t1, z0, z1, lon0, lon1, lat0, lat1;
00037
00038
00039
00040
          int i, ip, iq, n;
00041
           /* Allocate... */
00042
          ALLOC(atm, atm_t, 1);
ALLOC(atm2, atm_t, 1);
00043
00044
00045
00046
           /* Check arguments... */
00047
           if (argc < 4)
00048
             ERRMSG("Give parameters: <ctl> <atm_in> <atm_out>");
00049
00050
          /* Read control parameters... */
          /* Read control parameters... */
read_ctl(argv[1], argc, argv, &ctl);
n = (int) scan_ctl(argv[1], argc, argv, "SPLIT_N", -1, "", NULL);
m = scan_ctl(argv[1], argc, argv, "SPLIT_M", -1, "-999", NULL);
dt = scan_ctl(argv[1], argc, argv, "SPLIT_DT", -1, "0", NULL);
t0 = scan_ctl(argv[1], argc, argv, "SPLIT_TO", -1, "0", NULL);
t1 = scan_ctl(argv[1], argc, argv, "SPLIT_TI", -1, "0", NULL);
dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);
00051
00052
00053
00054
00055
00056
           dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);
00057
           z0 = scan_ctl(argv[1], argc, argv, "SPLIT_ZO", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "SPLIT_Z1", -1, "0", NULL);
dx = scan_ctl(argv[1], argc, argv, "SPLIT_DX", -1, "0", NULL);
00058
00059
00060
          lon0 = scan_ctl(argv[1], argc, argv, "SPLIT_LON0", -1, "0", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "SPLIT_LON1", -1, "0", NULL);
lat0 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT0", -1, "0", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT1", -1, "0", NULL);
00061
00062
00063
00064
00065
00066
           /* Init random number generator... */
00067
           gsl_rng_env_setup();
00068
           rng = gsl_rng_alloc(gsl_rng_default);
00069
00070
           /* Read atmospheric data... */
00071
           read_atm(argv[2], &ctl, atm);
00072
00073
           /* Get total and maximum mass... */
00074
           if (ctl.qnt_m >= 0)
            for (ip = 0; ip < atm->np; ip++) {
00075
                mtot += atm->q[ctl.qnt_m][ip];
00076
00077
                mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
00078
           if (m > 0)
00079
00080
             mtot = m:
00081
00082
           /* Loop over air parcels... */
00083
           for (i = 0; i < n; i++) {
00084
00085
              /* Select air parcel... */
00086
             if (ctl.qnt_m >= 0)
00087
                do {
00088
                  ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00089
                 } while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
```

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

# 5.31 time2jsec.c File Reference

Convert date to Julian seconds.

### **Functions**

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

## 5.31.1 Detailed Description

Convert date to Julian seconds.

Definition in file time2jsec.c.

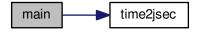
## 5.31.2 Function Documentation

## 5.31.2.1 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:



# 5.32 time2jsec.c

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

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

### 5.33 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.

• double module\_meteo\_hno3 (double t, double lat, double p)

Auxiliary function for meteo module.

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[])

## 5.33.1 Detailed Description

Lagrangian particle dispersion model.

Definition in file trac.c.

#### 5.33.2 Function Documentation

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

Set simulation time interval.

Definition at line 371 of file trac.c.

```
00373
00374
00375
          /\star Set inital and final time... \star/
00376
          if (ctl->direction == 1)
00377
            if (ctl->t_start < -1e99)
         ctl->t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
if (ctl->t_stop < -le99)
  ctl->t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
} else if (ctl->direction == -1) {
00378
00379
00380
00381
00382
           if (ctl->t_stop < -1e99)</pre>
00383
              ctl->t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00384
            if (ctl->t_start < -1e99)</pre>
               ctl->t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00385
00386
00387
          /* Check time... */
00389
          if (ctl->direction * (ctl->t_stop - ctl->t_start) <= 0)</pre>
00390
            ERRMSG("Nothing to do!");
00391 }
```

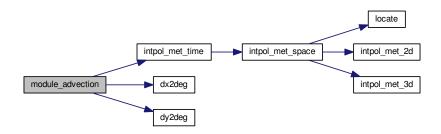
5.33.2.2 void module\_advection ( met\_t \* met0, met\_t \* met1, atm\_t \* atm, int ip, double dt )

Calculate advection of air parcels.

Definition at line 395 of file trac.c.

```
00400
                             {
00401
00402
            double v[3], xm[3];
00403
00404
            /* Interpolate meteorological data... */
            intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00405
00406
                                     atm->lon[ip], atm->lat[ip], NULL, NULL,
00407
                                     &v[0], &v[1], &v[2], NULL, NULL);
00408
            /* 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.);
00409
00410
00411
00412
            xm[2] = atm - p[ip] + 0.5 * dt * v[2];
00413
00414
            /\star Interpolate meteorological data for mid point... \star/
             \begin{array}{c} intpol\_met\_time \, (met0, \,\, met1, \,\, atm->time \, [ip] \,\, + \,\, 0.5 \,\, \star \,\, dt, \\ xm[2], \,\, xm[0], \,\, xm[1], \,\, NULL, \,\, NULL, \,\, & \\ & v[0], \,\, \&v[1], \,\, \&v[2], \,\, NULL, \,\, NULL) \,; \end{array} 
00415
00416
00417
00418
00419
            /* Save new position... */
00420
            atm->time[ip] += dt;
            atm->lan[ip] += dx2deg(dt * v[0] / 1000., xm[1]);
atm->lat[ip] += dy2deg(dt * v[1] / 1000.);
00421
00422
            atm->p[ip] += dt * v[2];
00423
00424 }
```

Here is the call graph for this function:



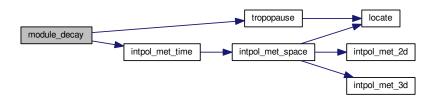
```
5.33.2.3 void module_decay ( ctl_t * ctl, met_t * met0, met_t * met1, atm_t * atm, int ip, double dt )
```

Calculate exponential decay of particle mass.

Definition at line 428 of file trac.c.

```
00434
00435
00436
        double ps, pt, tdec;
00437
        /* Check lifetime values... */
00439
         if ((ctl->tdec_trop <= 0 && ctl->tdec_strat <= 0) || ctl->
      qnt_m < 0)
00440
          return;
00441
00442
        /* Set constant lifetime... */
        if (ctl->tdec_trop == ctl->tdec_strat)
00444
          tdec = ctl->tdec_trop;
00445
00446
        /* Set altitude-dependent lifetime... */
00447
        else {
00448
           /* Get surface pressure... */
00450
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00451
                           atm->lon[ip], atm->lat[ip], &ps, NULL,
00452
                           NULL, NULL, NULL, NULL, NULL);
00453
00454
          /* Get tropopause pressure... */
00455
          pt = tropopause(atm->time[ip], atm->lat[ip]);
00456
00457
          /\star Set lifetime... \star/
00458
          if (atm->p[ip] \le pt)
00459
            tdec = ctl->tdec_strat;
00460
          else
00461
            tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
     p[ip]);
00462
00463
        /* Calculate exponential decay... */
atm->q[ctl->qnt_m][ip] *= exp(-dt / tdec);
00464
00465
00466 }
```

Here is the call graph for this function:



5.33.2.4 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 470 of file trac.c.

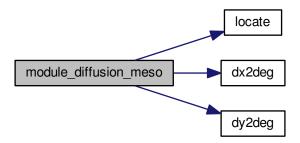
```
00477
00478
00479
         double r, rs, u[16], v[16], w[16], usig, vsig, wsig;
00480
00481
         int ix, iv, iz;
00482
00483
          /* Calculate mesoscale velocity fluctuations... */
00484
          if (ctl->turb_meso > 0) {
00485
00486
            /* Get indices... */
00487
           ix = locate(met0->lon, met0->nx, atm->lon[ip]);
            iy = locate(met0->lat, met0->ny, atm->lat[ip]);
00488
00489
            iz = locate(met0->p, met0->np, atm->p[ip]);
00490
00491
            /* Collect local wind data... */
           u[0] = met0->u[ix][iy][iz];
u[1] = met0->u[ix + 1][iy][iz];
00492
00493
           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];
00494
00496
            u[4] = met0 -> u[ix][iy][iz + 1];
00497
            u[5] = met0 -> u[ix + 1][iy][iz + 1];
00498
            u[6] = met0 -> u[ix][iy + 1][iz + 1];
            u[7] = met0 -> u[ix + 1][iy + 1][iz + 1];
00499
00500
00501
            v[0] = met0 -> v[ix][iy][iz];
            v[1] = met0 -> v[ix + 1][iy][iz];
00502
00503
            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];
00504
00505
            v[5] = met0->v[ix + 1][iy][iz + 1];
v[6] = met0->v[ix][iy + 1][iz + 1];
00506
00507
00508
            v[7] = met0 -> v[ix + 1][iy + 1][iz + 1];
00509
00510
            w[0] = met0->w[ix][iy][iz];
00511
            w[1] = met0->w[ix + 1][iy][iz];
            w[2] = met0->w[ix][iy + 1][iz];
00512
            w[3] = met0->w[ix + 1][iy + 1][iz];
w[4] = met0->w[ix][iy][iz + 1];
00513
00515
            w[5] = met0 -> w[ix + 1][iy][iz + 1];
00516
            w[6] = met0 -> w[ix][iy + 1][iz + 1];
00517
            w[7] = met0 -> w[ix + 1][iy + 1][iz + 1];
00518
            /* Get indices... */
00519
            ix = locate(met1->lon, met1->nx, atm->lon[ip]);
00520
            iy = locate(met1->lat, met1->ny, atm->lat[ip]);
00521
00522
            iz = locate(met1->p, met1->np, atm->p[ip]);
00523
00524
            /* Collect local wind data... */
           u[8] = met1->u[ix][iy][iz];
u[9] = met1->u[ix + 1][iy][iz];
00525
00526
            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];
00528
00529
           u[13] = met1->u[ix + 1][iy][iz + 1];

u[14] = met1->u[ix][iy + 1][iz + 1];
00530
00531
00532
            u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00534
            v[8] = met1->v[ix][iy][iz];
00535
            v[9] = met1 -> v[ix + 1][iy][iz];
00536
            v[10] = met1->v[ix][iy + 1][iz];
            v[13] = met1 > v[ix][iy + 1][iz];
v[11] = met1 - > v[ix + 1][iy + 1][iz];
v[12] = met1 - > v[ix][iy][iz + 1];
00537
00538
            v[13] = met1->v[ix + 1][iy][iz + 1];
v[14] = met1->v[ix][iy + 1][iz + 1];
00539
00540
00541
            v[15] = met1 - v[ix + 1][iy + 1][iz + 1];
00542
00543
            w[8] = met1->w[ix][iy][iz];
            w[9] = met1->w[ix + 1][iy][iz];
00544
            w[10] = met1->w[ix][iy + 1][iz];
00545
            w[11] = met1->w[ix][iy + 1][iz];
w[11] = met1->w[ix + 1][iy + 1][iz];
w[12] = met1->w[ix][iy][iz + 1];
00546
00547
00548
            w[13] = met1->w[ix + 1][iy][iz + 1];
            w[14] = met1->w[ix][iy + 1][iz + 1];
w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00549
00550
00551
00552
            /* Get standard deviations of local wind data... */
00553
            usig = gsl_stats_sd(u, 1, 16);
00554
            vsig = gsl_stats_sd(v, 1, 16);
00555
            wsig = gsl_stats_sd(w, 1, 16);
00556
00557
            /\star Set temporal correlations for mesoscale fluctuations... \star/
            r = 1 - 2 * fabs(dt) / ctl->dt_met;
00559
            rs = sqrt(1 - r * r);
00560
00561
            /\!\star Calculate mesoscale wind fluctuations... \star/
00562
            atm->up[ip] = (float)
00563
              (r * atm->up[ip]
```

5.33 trac.c File Reference 175

```
+ rs * qsl_ran_qaussian_ziqqurat(rnq, ctl->turb_meso * usiq));
00565
            atm->vp[ip] = (float)
            (r * atm->vp[ip]
00566
00567
               + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_meso * vsig));
00568
            atm->wp[ip] = (float)
            (r * atm->wp[ip]
00569
00570
               + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_meso * wsig));
00571
           /* 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.);
00572
00573
00574
00575
            atm->p[ip] += atm->wp[ip] * dt;
00576
00577 }
```

Here is the call graph for this function:



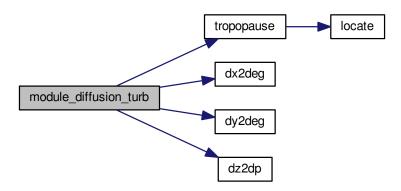
5.33.2.5 void module diffusion turb (ctl t\*ctl, atm t\*atm, int ip, double dt, gsl rng \* rng)

Calculate turbulent diffusion.

Definition at line 581 of file trac.c.

```
00586
00587
00588
        double dx, dz, pt, p0, p1, w;
00589
00590
        /* Get tropopause pressure... */
00591
        pt = tropopause(atm->time[ip], atm->lat[ip]);
00592
00593
        /* Get weighting factor... */
        p1 = pt * 0.866877899;
p0 = pt / 0.866877899;
00594
00595
00596
        if (atm->p[ip] > p0)
00597
        else if (atm->p[ip] < p1)</pre>
00598
          w = 0;
00599
00600
        else
          w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00601
00602
         /* Set diffusivitiy... */
        dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00604
00605
00606
00607
        /* Horizontal turbulent diffusion... */
00608
        if (dx > 0) {
00609
          atm->lon[ip]
00610
             += dx2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00611
                         / 1000., atm->lat[ip]);
          atm->lat[ip]
00612
00613
             += dy2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00614
                        / 1000.);
00615
```

Here is the call graph for this function:



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

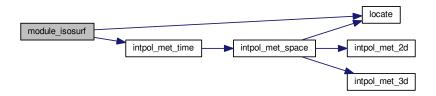
Force air parcels to stay on isosurface.

Definition at line 626 of file trac.c.

```
00631
                  {
00632
         static double *iso, *ps, t, *ts;
00634
00635
        static int idx, ip2, n;
00636
00637
        FILE *in;
00638
00639
        char line[LEN];
00640
         /* Check control parameter... */ if (ctl->isosurf < 1 || ctl->isosurf > 4)
00641
00642
00643
           return:
00644
00645
         /* Initialize... */
00646
         if (ip < 0) {</pre>
00647
00648
           /* Allocate... */
00649
           ALLOC(iso, double,
                  NP);
00650
           ALLOC(ps, double, NP);
00651
00652
00653
           ALLOC(ts, double,
00654
                  NP);
00655
           /* Save pressure... */
if (ctl->isosurf == 1)
00656
00657
00658
             for (ip2 = 0; ip2 < atm->np; ip2++)
00659
                iso[ip2] = atm->p[ip2];
00660
00661
           /* Save density... */
else if (ctl->isosurf == 2)
00662
00663
             for (ip2 = 0; ip2 < atm->np; ip2++) {
00664
                intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
```

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

Here is the call graph for this function:



```
5.33.2.7 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 741 of file trac.c.

```
00746
                 {
00747
00748
        static FILE *in;
00749
00750
        static char filename[LEN], line[LEN];
00751
        static double lon[GX], lat[GY], var[GX][GY],
rdum, rlat, rlat_old = -999, rlon, rvar;
00752
00753
00754
00755
        static int year_old, mon_old, day_old, nlon, nlat;
00756
00757
        double a, b, c, ps, p1, p_hno3, p_h2o, t, t1, u, u1, v, v1, w,
00758
          x1, x2, h2o, o3, grad, vort, var0, var1;
00759
00760
        int day, mon, year, idum, ilat, ilon;
00761
00762
        /* Interpolate meteorological data... */
00763
        intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00764
                         atm->lat[ip], &ps, &t, &u, &v, &w, &h2o, &o3);
00765
00766
        /* Set surface pressure... */
        if (ctl->qnt_ps >= 0)
00767
00768
          atm->q[ctl->qnt_ps][ip] = ps;
00769
00770
        /* Set pressure... */
if (ctl->qnt_p >= 0)
00771
00772
          atm->q[ctl->qnt_p][ip] = atm->p[ip];
00773
00774
        /* Set temperature... */
        if (ctl->qnt_t >= 0)
00775
00776
          atm->q[ctl->qnt_t][ip] = t;
00777
00778
        /* Set zonal wind... */
00779
        if (ctl->qnt_u >= 0)
00780
          atm->q[ctl->qnt_u][ip] = u;
00781
00782
        /\star Set meridional wind... \star/
00783
        if (ctl->qnt_v >= 0)
          atm->q[ctl->qnt_v][ip] = v;
00784
00785
00786
        /\star Set vertical velocity... \star/
00787
        if (ctl->qnt_w >= 0)
00788
          atm->q[ctl->qnt_w][ip] = w;
00789
00790
        /* Set water vapor vmr... */
00791
        if (ct1->qnt_h2o>=0)
00792
          atm->q[ctl->qnt_h2o][ip] = h2o;
00793
00794
        /* Set ozone vmr... */
00795
        if (ctl->qnt o3 >= 0)
          atm->q[ctl->qnt_o3][ip] = o3;
00796
00797
00798
        /* Calculate potential temperature... */
```

```
if (ctl->qnt_theta >= 0)
00800
          atm \rightarrow q[ctl \rightarrow qnt\_theta][ip] = t * pow(P0 / atm \rightarrow p[ip], 0.286);
00801
00802
         /* Calculate potential vorticity... */
00803
         if (ctl->qnt_pv >= 0) {
00804
            /* Absolute vorticity... */
00806
           vort = 2 * 7.2921e-5 * sin(atm->lat[ip] * M_PI / 180.);
00807
           if (fabs(atm->lat[ip]) < 89.) {
00808
              intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
                                (atm->lon[ip] >=
00809
00810
                                 0 ? atm->lon[ip] - 1. : atm->lon[ip] + 1.),
00811
                                atm->lat[ip], NULL, NULL, NULL, &v1, NULL, NULL, NULL);
              vort += (v1 - v) / 1000.
00812
00813
               / ((atm->lon[ip] >= 0 ? -1 : 1) * deg2dx(1., atm->lat[ip]));
00814
00815
           intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00816
                              (atm->lat[ip] >=
00817
                               0 ? atm->lat[ip] - 1. : atm->lat[ip] + 1.), NULL, NULL,
           &u1, NULL, NULL, NULL);

vort += (u1 - u) / 1000. / ((atm->lat[ip] >= 0 ? -1 : 1) * deg2dy(1.));
00818
00819
00820
00821
           /* Potential temperature gradient... */
00822
           p1 = 0.85 * atm->p[ip];
           intpol_met_time(met0, met1, atm->time[ip], p1, atm->lon[ip],
00824
                             atm->lat[ip], NULL, &t1, NULL, NULL, NULL, NULL, NULL);
00825
           grad = (t1 * pow(P0 / p1, 0.286) - t * pow(P0 / atm->p[ip], 0.286))
             / (100. * (p1 - atm->p[ip]));
00826
00827
00828
           /* Calculate PV... */
00829
           atm \rightarrow q[ctl \rightarrow qnt_pv][ip] = -1e6 * G0 * vort * grad;
00830
00831
         /* Calculate T_ice (Marti and Mauersberger, 1993)... */
if (ctl->qnt_tice >= 0 || ctl->qnt_tsts >= 0)
00832
00833
          atm->q[ctl->qnt_tice][ip] =
00834
             -2663.5 /
00836
              (log10((ct1->psc_h2o > 0 ? ct1->psc_h2o : h2o) * atm->p[ip] * 100.) -
00837
00838
00839
         /* Calculate T NAT (Hanson and Mauersberger, 1988)... */
         if (ctl->qnt_tnat >= 0 || ctl->qnt_tsts >= 0) {
   if (ctl->psc_hno3 > 0)
00840
00841
00842
             p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
00843
00844
             p_hno3 = module_meteo_hno3(atm->time[ip], atm->lat[ip], atm->
      p[ip])
00845
           * le-9 * atm->p[ip] / 1.333224;

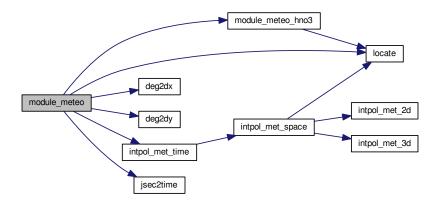
p_h2o = (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;

a = 0.009179 - 0.00088 * log10(p_h2o);
00846
00848
           b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
           c = -11397.0 / a;
00849
           x1 = (-b + sqrt(b * b - 4. * c)) / 2.;

x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
00850
00851
           if (x1 > 0)
00852
             atm->q[ctl->qnt_tnat][ip] = x1;
00854
           if (x2 > 0)
00855
             atm->q[ctl->qnt_tnat][ip] = x2;
00856
00857
00858
         /* Calculate T_STS (mean of T_ice and T_NAT)... */
00859
         if (ctl->qnt_tsts >= 0) {
          if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)</pre>
00860
00861
             ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
00862
           atm -> q[ctl -> qnt\_tsts][ip] = 0.5 * (atm -> q[ctl -> qnt\_tice][ip]
00863
                                                    + atm->q[ctl->qnt_tnat][ip]);
00864
00865
         /* Read variance data for current day... */
00867
         if (ip == 0 && ctl->qnt_gw_var >= 0) {
           jsec2time(atm->time[ip], &year, &mon, &day, &idum, &idum, &idum, &rdum);
00868
           if (year != year_old || mon != mon_old || day != day_old) {
  year_old = year;
00869
00870
             mon_old = mon;
day_old = day;
00871
00872
00873
             nlon = nlat = -1;
00874
             sprintf(filename, "%s_%d_%02d_%02d.tab",
             ctl->gw_basename, year, mon, day);
if ((in = fopen(filename, "r"))) {
00875
00876
00877
               printf("Read gravity wave data: %s\n", filename);
                while (fgets(line, LEN, in)) {
  if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rvar) != 3)
00878
00879
00880
                     continue;
                  if (rlat != rlat_old) {
  rlat_old = rlat;
00881
00882
00883
                    if ((++nlat) > GY)
```

```
ERRMSG("Too many latitudes!");
00885
                 nlon = -1;
00886
00887
               if ((++nlon) > GX)
                 ERRMSG("Too many longitudes!");
00888
               lon[nlon] = rlon;
lat[nlat] = rlat;
00889
00890
00891
               var[nlon][nlat] = GSL_MAX(0, rvar);
00892
00893
             fclose(in);
00894
             nlat++;
00895
             nlon++;
00896
           } else
00897
             printf("Warning: Missing gravity wave data: %s\n", filename);
00898
00899
00900
00901
       /* Interpolate variance data... */
       if (ctl->qnt_gw_var >= 0) {
00902
00903
         if (nlat >= 2 && nlon >= 2) {
00904
           ilat = locate(lat, nlat, atm->lat[ip]);
           ilon = locate(lon, nlon, atm->lon[ip]);
00905
           00906
00907
00908
00909
00910
           atm->q[ctl->qnt_gw_var][ip]
00911
             = LIN(lon[ilon], var0, lon[ilon + 1], var1, atm->lon[ip]);
         } else
00912
00913
           atm->q[ctl->qnt\_gw\_var][ip] = GSL\_NAN;
00914
00915 }
```

Here is the call graph for this function:



## 5.33.2.8 double module\_meteo\_hno3 ( double t, double lat, double p )

Auxiliary function for meteo module.

Definition at line 919 of file trac.c.

```
00922
00923
        static double secs[12] = { 1209600.00, 3888000.00, 6393600.00,
00924
00925
          9072000.00, 11664000.00, 14342400.00,
00926
          16934400.00, 19612800.00, 22291200.00,
00927
         24883200.00, 27561600.00, 30153600.00
00928
00929
00930
        static double lats[18] = { -85, -75, -65, -55, -45, -35, -25, -15, -5,
00931
         5, 15, 25, 35, 45, 55, 65, 75, 85
```

```
00932
00933
00934
          static double ps[10] = { 4.64159, 6.81292, 10, 14.678, 21.5443,
00935
            31.6228, 46.4159, 68.1292, 100, 146.78
00936
00937
         static double hno3[12][18][10] = {
            {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00939
00940
              {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00941
              {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
              {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00942
00943
              \{0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709\}
              {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.371, {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00944
00945
00946
              {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
              {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181}, {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104}, {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985}, {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00947
00948
00949
              \{0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745\},
              {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00952
00953
              {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
              {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97}, {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14}, {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
00954
00955
00956
            {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
              {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
00958
              {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33}, {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00959
00960
              {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
00961
00962
              {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
              {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
              {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
00964
00965
              {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167}
              {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101}, {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00966
00967
              {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191}, {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
00968
00970
              \{0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11\},
00971
              {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
              {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64}, {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
00972
00973
            {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
{{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00974
              \{0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52\},
00976
00977
              {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
              {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98}
{0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
00978
00979
              \{0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33\},
00980
              {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
00981
              \{0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169\},
              {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
00983
00984
              {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121}
00985
              \{0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135\},
              {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194}, {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
00986
00987
              {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
{0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
00989
00990
              {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
00991
              {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08}
00992
              \{1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42\}\},
            {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
00993
              {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
              {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
00995
00996
              {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09}
00997
              {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727}
00998
              {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409}, {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
00999
              {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12}
01000
              \{0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172\},
              {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
01002
01003
              {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
              {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286}, {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
01004
01005
01006
              \{0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96\}
              {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
              {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
01008
01009
              {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46}
              {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
01010
            {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
01011
              {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57}, {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
01012
              {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
01014
01015
              {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88}
01016
              {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
              {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229}, {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
01017
01018
```

```
\{0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126\},\
              \{0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183\},
01020
01021
              {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
              {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
01022
              {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
01023
              {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
01024
              {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
              {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
01026
01027
              {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
            {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}}, {1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
01028
01029
              \{1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05\},\
01030
              {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
              {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
01032
01033
              {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
              {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639}, {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
01034
01035
              {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
01036
              {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
              \{0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194\},
01038
              {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
01039
01040
              {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
              \{0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66\},
01041
              {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41}, {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
01042
01043
              {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9}
01045
              {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
              {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}}
01046
01047
            \{\{3.58,\ 2.59,\ 6.49,\ 5.84,\ 1.63,\ 0.282,\ 0.647,\ 0.371,\ 1.36,\ 2.33\},
01048
              {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
              {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
01049
              {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
              {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
01051
01052
              {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
              {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176}, {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
01053
01054
              {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12}, {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
01055
01057
              {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25}
              \{0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259\},
01058
01059
              {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
              {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
01060
01061
              {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
01062
              {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
              \{0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61\},
01063
01064
              {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}},
            {5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4}, {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73}, {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6}, {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
01065
01066
01067
01068
              (0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38), (0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672),
01070
01071
              {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19}
              {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181}, {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
01072
01073
              {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
01074
              {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
              {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232}
01076
              {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
01077
01078
              \{0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754\},
01079
              \{0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23\},
              {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45}, {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
01080
              \{0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55\}
01083
             {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
01084
              {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
01085
              {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09}, {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
01086
              (0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22), (0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646),
01087
              {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169}, {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
01089
01090
01091
              {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
              {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
01092
              {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
{1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
01093
              \{0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303\},
01095
01096
              {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714}, {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1}, {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
01097
01098
              {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56}, {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
01099
             {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
01101
01102
              {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
01103
              {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
              {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56}, {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
01104
01105
```

```
{0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
                \{0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21\},
01108
                {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
                {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
01109
                {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146}, {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
01110
01111
                {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
01112
                {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
01113
01114
                {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
                {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2}, {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
01115
01116
                {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
01117
                {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
01118
              {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
01119
01120
                {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
01121
                \{0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75\},
                {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
01122
                \{0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955\},
01123
                \{0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61\},
                {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
                {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
01126
01127
                {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
                {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147}, {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146}, {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172}, {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448}, {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
01128
01129
01130
01131
01132
01133
                {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
                {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76}, {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97}, {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
01134
01135
01136
              {0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89}, {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
01137
01138
               {0.699, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74}, {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65}, {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28}, {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837}, {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488}, {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256}, {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
01139
01140
01141
01142
                {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173}, {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138}, {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133}, {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189}, {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
01145
01146
01147
01148
01149
                {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
01150
01151
                    24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
01152
                {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
                {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35}, {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
01153
01154
01155
01156
01157
           double aux00, aux01, aux10, aux11, sec;
01158
01159
           int ilat, ip, isec;
01160
01161
           /* Get seconds since begin of year... */
           sec = fmod(t, 365.25 * 86400.);
01163
01164
            /* Get indices...
01165
           ilat = locate(lats, 18, lat);
01166
           ip = locate(ps, 10, p);
01167
           isec = locate(secs, 12, sec);
01168
01169
           /* Interpolate...
01170
           aux00 = LIN(ps[ip], hno3[isec][ilat][ip],
01171
                            ps[ip + 1], hno3[isec][ilat][ip + 1], p);
01172
           aux01 = LIN(ps[ip], hno3[isec][ilat + 1][ip],
                             ps[ip + 1], hno3[isec][ilat + 1][ip + 1], p);
01173
           aux10 = LIN(ps[ip], hno3[isec + 1][ilat][ip],
01174
                             ps[ip + 1], hno3[isec + 1][ilat][ip + 1], p);
01176
           aux11 = LIN(ps[ip], hno3[isec + 1][ilat + 1][ip],
01177
                             ps[ip + 1], hno3[isec + 1][ilat + 1][ip + 1], p);
           aux00 = LIN(lats[ilat], aux00, lats[ilat + 1], aux01, lat);
aux11 = LIN(lats[ilat], aux10, lats[ilat + 1], aux11, lat);
01178
01179
01180
           return LIN(secs[isec], aux00, secs[isec + 1], aux11, sec);
```

Here is the call graph for this function:



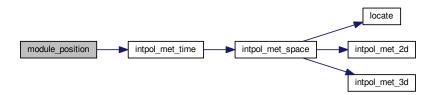
5.33.2.9 void module\_position ( met\_t \* met0, met\_t \* met1, atm\_t \* atm, int ip )

Check position of air parcels.

Definition at line 1185 of file trac.c.

```
01189
01190
01191
          double ps;
01192
          /* Calculate modulo... */
          atm->lon[ip] = fmod(atm->lon[ip], 360);
atm->lat[ip] = fmod(atm->lat[ip], 360);
01194
01195
01196
          /* Check latitude... */
while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
01197
01198
           if (atm->lat[ip] > 90) {
  atm->lat[ip] = 180 - atm->lat[ip];
  atm->lon[ip] += 180;
01199
01200
01201
01202
            if (atm->lat[ip] < -90) {
  atm->lat[ip] = -180 - atm->lat[ip];
  atm->lon[ip] += 180;
01203
01204
01205
01206
01207
01208
          /* Check longitude... */
while (atm->lon[ip] < -180)
atm->lon[ip] += 360;
01209
01210
01211
01212
          while (atm->lon[ip] >= 180)
01213
            atm->lon[ip] -= 360;
01214
          01215
01216
01217
01218
                               NULL, NULL, NULL, NULL, NULL);
01219
01220
          /* Check pressure... */
          if (atm->p[ip] > ps)
01221
          atm->p[ip] = ps;
else if (atm->p[ip] < met0->p[met0->np - 1])
atm->p[ip] = met0->p[met0->np - 1];
01222
01223
01224
01225 }
```

Here is the call graph for this function:



5.33 trac.c File Reference 185

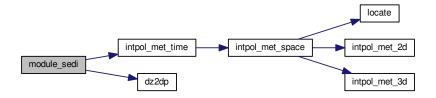
5.33.2.10 void module\_sedi ( ctl\_t \* ctl, met\_t \* met0, met\_t \* met1, atm\_t \* atm, int ip, double dt )

Calculate sedimentation of air parcels.

Definition at line 1229 of file trac.c.

```
01235
                     {
01236
         /\star Coefficients for Cunningham slip-flow correction (Kasten, 1968): \star/
01238
        const double A = 1.249, B = 0.42, C = 0.87;
01239
        /* Specific gas constant for dry air [J/(kg K)]: \star/ const double R = 287.058;
01240
01241
01242
01243
         /* Average mass of an air molecule [kg/molec]: */
01244
        const double m = 4.8096e-26;
01245
01246
        double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
01247
01248
        /\star Check if parameters are available... \star/
01249
        if (ctl->qnt_r < 0 || ctl->qnt_rho < 0)</pre>
01250
          return;
01251
01252
         /* Convert units... */
01253
        p = 100 * atm->p[ip];
        r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
01254
01255
        rho_p = atm->q[ctl->qnt_rho][ip];
01256
01257
         /* Get temperature... */
        intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
01258
      lon[ip],
01259
                          atm->lat[ip], NULL, &T, NULL, NULL, NULL, NULL, NULL);
01260
01261
        /* Density of dry air... */
01262
        rho = p / (R * T);
01263
01264
        /* Dynamic viscosity of air... */
        eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
01265
01266
01267
        /* Thermal velocity of an air molecule... */ v = sqrt(8 * GSL_CONST_MKSA_BOLTZMANN * T / (M_PI * m));
01268
01269
01270
         /\star Mean free path of an air molecule... \star/
01271
        lambda = 2 * eta / (rho * v);
01272
01273
         /* Knudsen number for air... */
01274
        K = lambda / r_p;
01275
01276
        /* Cunningham slip-flow correction... */
01277
        G = 1 + K * (A + B * exp(-C / K));
01278
01279
         /* Sedimentation (fall) velocity... */
01280
01281
           2. * gsl_pow_2(r_p) * (rho_p
01282
                                    rho) * GSL_CONST_MKSA_GRAV_ACCEL / (9. * eta) * G;
01283
        /* Calculate pressure change... */
atm->p[ip] += dz2dp(v_p * dt / 1000., atm->p[ip]);
01284
01285
01286 }
```

Here is the call graph for this function:



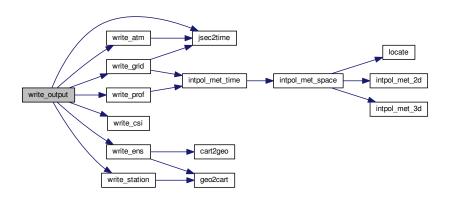
5.33.2.11 void write\_output ( const char \* dirname, ctl\_t \* ctl, met\_t \* met0, met\_t \* met1, atm\_t \* atm, double t )

Write simulation output.

Definition at line 1290 of file trac.c.

```
01296
                     {
01297
01298
         char filename[LEN];
01299
01300
         double r;
01301
01302
         int year, mon, day, hour, min, sec;
01303
01304
         /* Get time... */
01305
         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01306
         /* Write atmospheric data... */
if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01307
01308
         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab", dirname, ctl->atm_basename, year, mon, day, hour, min);
01309
01310
01311
           write_atm(filename, ctl, atm, t);
01312
01313
01314
         /* Write CST data... */
         if (ctl->csi_basename[0] != '-') {
01315
           sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01316
01317
           write_csi(filename, ctl, atm, t);
01318
01319
         /* Write ensemble data... */
if (ctl->ens_basename[0] != '-') {
01320
01321
         sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01322
01323
           write_ens(filename, ctl, atm, t);
01324
01325
         /* Write gridded data... */
if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01326
01327
01328
         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01329
                     dirname, ctl->grid_basename, year, mon, day, hour, min);
01330
           write_grid(filename, ctl, met0, met1, atm, t);
01331
01332
         /* Write profile data... */
if (ctl->prof_basename[0] != '-') {
   sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
   write_prof(filename, ctl, met0, met1, atm, t);
01333
01334
01336
01337
01338
         /* Write station data...
01339
         if (ctl->stat_basename[0] != '-') {
01340
         sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01341
01342
           write_station(filename, ctl, atm, t);
01343
01344 }
```

Here is the call graph for this function:



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

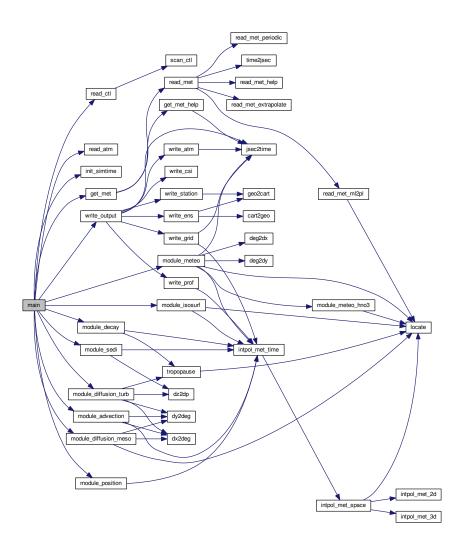
Definition at line 126 of file trac.c.

```
00128
00129
00130
        ctl t ctl:
00131
00132
        atm_t *atm;
00133
00134
        met_t *met0, *met1;
00135
00136
        gsl_rng *rng[NTHREADS];
00137
00138
        FILE *dirlist;
00139
00140
        char dirname[LEN], filename[LEN];
00141
00142
        double *dt. t. t0:
00143
00144
        int i, ip, ntask = 0, rank = 0, size = 1;
00145
00146 #ifdef MPI
00147
        /* Initialize MPI... */
00148
        MPI_Init(&argc, &argv);
        MPI_Comm_rank (MPI_COMM_WORLD, &rank);
00149
00150
        MPI_Comm_size(MPI_COMM_WORLD, &size);
00151 #endif
00152
00153
         /* Check arguments... */
        if (argc < 5)
00154
          ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00155
00156
00157
        /* Open directory list... */
00158
        if (!(dirlist = fopen(argv[1], "r")))
00159
          ERRMSG("Cannot open directory list!");
00160
        /* Loop over directories... */
while (fscanf(dirlist, "%s", dirname) != EOF) {
00161
00162
00163
00164
           /* MPI parallelization... */
00165
          if ((++ntask) % size != rank)
             continue;
00166
00167
00168
00169
             Initialize model run...
00170
00171
00172
           /* Set timers... */
          START_TIMER(TIMER_TOTAL);
00173
00174
          START_TIMER(TIMER_INIT);
00175
00176
           /* Allocate... */
00177
           ALLOC(atm, atm_t, 1);
00178
           ALLOC(met0, met_t, 1);
00179
          ALLOC(met1, met_t, 1);
          ALLOC(dt, double, NP);
00180
00181
00182
          /* Read control parameters... */
sprintf(filename, "%s/%s", dirname, argv[2]);
00183
00184
00185
           read_ctl(filename, argc, argv, &ctl);
00186
00187
           /* Initialize random number generators... */
00188
          gsl_rng_env_setup();
00189
          if (omp_get_max_threads() > NTHREADS)
00190
            ERRMSG("Too many threads!");
           for (i = 0; i < NTHREADS; i++)
rng[i] = gsl_rng_alloc(gsl_rng_default);</pre>
00191
00192
00193
          /* Read atmospheric data... */
sprintf(filename, "%s/%s", dirname, argv[3]);
00194
00195
00196
           read_atm(filename, &ctl, atm);
00197
00198
           /* Get simulation time interval... */
00199
           init_simtime(&ctl, atm);
00200
00201
           /* Get rounded start time... */
00202
           if (ctl.direction == 1)
00203
            t0 = floor(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00204
          else
00205
            t0 = ceil(ctl.t start / ctl.dt mod) * ctl.dt mod;
00206
           /* Set timers... */
```

```
STOP_TIMER(TIMER_INIT);
00209
00210
00211
             Loop over timesteps...
00212
00213
00214
          /* Loop over timesteps... */
00215
          for (t = t0; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;</pre>
00216
                t += ctl.direction * ctl.dt_mod) {
00217
00218
             /* Adjust length of final time step... */
            if (ctl.direction * (t - ctl.t_stop) > 0)
00219
00220
              t = ctl.t stop;
00221
00222
             /\star Set time steps for air parcels... \star/
00223
             for (ip = 0; ip < atm\rightarrownp; ip++)
               if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
00224
                   && ctl.direction * (atm->time[ip] - ctl.t_stop) <= 0
&& ctl.direction * (atm->time[ip] - t) < 0))
00225
00227
                dt[ip] = t - atm->time[ip];
00228
00229
                dt[ip] = GSL_NAN;
00230
00231
             /* Get meteorological data... */
00232
             START_TIMER(TIMER_INPUT);
00233
            get_met(&ctl, argv[4], t, met0, met1);
00234
             if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
              printf("Warning: Violation of CFL criterion! Set DT_MOD <= %g s!\n",</pre>
00235
                      fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.);
00236
            STOP_TIMER(TIMER_INPUT);
00237
00238
00239
             /* Initialize isosurface... */
00240
             START_TIMER(TIMER_ISOSURF);
             if (t == t0)
00241
              module_isosurf(&ctl, met0, met1, atm, -1);
00242
00243
             STOP_TIMER(TIMER_ISOSURF);
00244
             /* Advection...
00246
            START_TIMER(TIMER_ADVECT);
00247 #pragma omp parallel for default(shared) private(ip)
00248
            for (ip = 0; ip < atm->np; ip++)
             if (gsl_finite(dt[ip]))
00249
00250
                module advection(met0, met1, atm, ip, dt[ip]);
00251
            STOP_TIMER(TIMER_ADVECT);
00252
00253
             /* Turbulent diffusion..
00254
            START_TIMER(TIMER_DIFFTURB);
00255 #pragma omp parallel for default(shared) private(ip)
            for (ip = 0; ip < atm->np; ip++)
00256
              if (gsl_finite(dt[ip]))
00257
                module_diffusion_turb(&ctl, atm, ip, dt[ip],
00259
                                        rng[omp_get_thread_num()]);
00260
            STOP_TIMER(TIMER_DIFFTURB);
00261
00262
             /* Mesoscale diffusion...
00263
            START_TIMER(TIMER_DIFFMESO);
00264 #pragma omp parallel for default(shared) private(ip)
            for (ip = 0; ip < atm->np; ip++)
00265
00266
             if (gsl_finite(dt[ip]))
00267
                module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
00268
                                        rng[omp_get_thread_num()]);
00269
            STOP TIMER (TIMER DIFFMESO);
00270
00271
             /* Sedimentation...
00272
            START_TIMER (TIMER_SEDI);
00273 #pragma omp parallel for default(shared) private(ip)
00274
            for (ip = 0; ip < atm->np; ip++)
             if (gsl_finite(dt[ip]))
  module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
00275
00276
            STOP_TIMER(TIMER_SEDI);
00278
00279
             /* Isosurface... */
            START_TIMER(TIMER_ISOSURF);
00280
00281 #pragma omp parallel for default(shared) private(ip)
            for (ip = 0; ip < atm->np; ip++)
module_isosurf(&ctl, met0, met1, atm, ip);
00282
00283
00284
             STOP_TIMER(TIMER_ISOSURF);
00285
             /* Position...
00286
            START_TIMER(TIMER_POSITION);
00287
00288 #pragma omp parallel for default(shared) private(ip)
            for (ip = 0; ip < atm->np; ip++)
  module_position(met0, met1, atm, ip);
00289
00290
00291
            STOP_TIMER(TIMER_POSITION);
00292
             /* Meteorological data... */
00293
00294
            START_TIMER (TIMER_METEO);
```

```
module_meteo(&ctl, met0, met1, atm, 0);
00296 #pragma omp parallel for default(shared) private(ip)
          for (ip = 1; ip < atm->np; ip++)
    module_meteo(&ctl, met0, met1, atm, ip);
00297
00298
00299
              STOP_TIMER(TIMER_METEO);
00300
               /* Decay... *,
00302
              START_TIMER(TIMER_DECAY);
00303 #pragma omp parallel for default(shared) private(ip)
              for (ip = 0; ip < atm->np; ip++)
  if (gsl_finite(dt[ip]))
00304
00305
00306
                   module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00307
              STOP_TIMER(TIMER_DECAY);
00308
00309
               /∗ Write output...
00310
              START_TIMER(TIMER_OUTPUT);
00311
               write_output(dirname, &ctl, met0, met1, atm, t);
               STOP_TIMER(TIMER_OUTPUT);
00312
00313
00314
00315
00316
               Finalize model run...
00317
00318
00319
            /* Report timers...
00320
            STOP_TIMER(TIMER_TOTAL);
00321
            PRINT_TIMER(TIMER_TOTAL);
00322
            PRINT_TIMER(TIMER_INIT);
00323
            PRINT_TIMER (TIMER_STAGE);
            PRINT_TIMER(TIMER_INPUT);
PRINT_TIMER(TIMER_OUTPUT);
00324
00325
00326
            PRINT_TIMER (TIMER_ADVECT);
00327
            PRINT_TIMER(TIMER_DECAY);
00328
            PRINT_TIMER(TIMER_DIFFMESO);
00329
            PRINT_TIMER(TIMER_DIFFTURB);
00330
            PRINT_TIMER (TIMER_ISOSURF);
            PRINT_TIMER(TIMER_METEO);
PRINT_TIMER(TIMER_POSITION);
00331
00332
00333
            PRINT_TIMER (TIMER_SEDI);
00334
            /* Report memory usage... */
printf("MEMORY_ATM = %g MByte\n", sizeof(atm_t) / 1024. / 1024.);
printf("MEMORY_METEO = %g MByte\n", 2. * sizeof(met_t) / 1024. / 1024.);
printf("MEMORY_DYNAMIC = %g MByte\n",
00335
00336
00337
00338
00339
                     4 * NP * sizeof(double) / 1024. / 1024.);
00340
            printf("MEMORY_STATIC = g MByten",
                    ((3 * GX * GY + 4 * GX * GY * GZ) * sizeof(double)
+ (EX * EY + EX * EY * EP) * sizeof(float)
+ (GX * GY + GX * GY * GZ) * sizeof(int)) / 1024. / 1024.);
00341
00342
00343
00344
            /* Report problem size... */
printf("SIZE_NP = %d\n", atm->np);
00345
00346
00347
            printf("SIZE_TASKS = %d\n", size);
00348
            printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00349
00350
            /* Free random number generators... */
00351
            for (i = 0; i < NTHREADS; i++)</pre>
00352
              gsl_rng_free(rng[i]);
00353
            /* Free... */
00354
00355
            free(atm):
00356
            free (met0);
00357
            free (met1);
00358
            free(dt);
00359
00360
00361 #ifdef MPI
         /* Finalize MPI... */
00362
00363
         MPI_Finalize();
00364 #endif
00365
00366
         return EXIT_SUCCESS;
00367 }
```

Here is the call graph for this function:



```
00001 /*
           This file is part of MPTRAC.
00002
00003
           MPTRAC is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by
00004
00005
00006
           the Free Software Foundation, either version 3 of the License, or
00007
           (at your option) any later version.
00008
           MPTRAC is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00009
00010
00011
00012
           GNU General Public License for more details.
00013
           You should have received a copy of the GNU General Public License along with MPTRAC. If not, see <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
00014
00015
00016
00017
           Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 #ifdef MPI
00028 #include "mpi.h"
00029 #endif
00030
```

```
00032
         Functions...
00033
00034
00036 void init_simtime(
        ctl_t * ctl,
atm_t * atm);
00037
00039
00041 void module_advection(
00042
         met_t * met0,
00043
         met_t * met1,
         atm_t * atm,
00044
         int ip,
00045
00046
         double dt);
00047
00049 void module_decay(
        ctl_t * ctl,
met_t * met0,
met_t * met1,
atm_t * atm,
00050
00051
00052
00053
00054
         int ip,
00055
         double dt);
00056
00058 void module_diffusion_meso(
         ctl_t * ctl,
met_t * met0,
00059
00060
00061
         met_t * met1,
00062
         atm_t * atm,
00063
         int ip,
         double dt,
00064
00065
         gsl_rng * rng);
00066
00068 void module_diffusion_turb(
        ctl_t * ctl,
atm_t * atm,
00069
00070
00071
         int ip,
00072
         double dt,
00073
         gsl_rng * rng);
00074
00076 void module_isosurf(
        ctl_t * ctl,
met_t * met0,
00077
00078
00079
         met_t * met1,
atm_t * atm,
08000
00081
         int ip);
00082
00084 void module_meteo(
00085
        ctl_t * ctl,
00086
         met_t * met0,
         met_t * met1,
atm_t * atm,
00087
00088
00089
         int ip);
00090
00092 double module_meteo_hno3(
        double t,
double lat,
00093
00094
00095
         double p);
00096
00098 void module_position(
00099
         met_t * met0,
         met_t * met1,
atm_t * atm,
00100
00101
00102
         int ip);
00103
00105 void module_sedi(
00106 ctl_t * ctl,
00107 met_t * met0,
         met_t * met1,
atm_t * atm,
00108
00109
00110
         int ip,
00111
         double dt);
00112
00114 void write_output(
00115
         const char *dirname,
         ctl_t * ctl,
met_t * met0,
00116
00117
        met_t * met1,
atm_t * atm,
00118
00119
00120
         double t);
00121
00122 /* --
00123
         Main...
00124
00125
00126 int main(
00127 int argc,
00128
        char *argv[]) {
```

```
00129
00130
        ctl_t ctl;
00131
00132
        atm_t *atm;
00133
00134
        met t *met0, *met1;
00135
00136
        gsl_rng *rng[NTHREADS];
00137
00138
        FILE *dirlist;
00139
00140
        char dirname[LEN], filename[LEN];
00141
00142
        double *dt, t, t0;
00143
00144
        int i, ip, ntask = 0, rank = 0, size = 1;
00145
00146 #ifdef MPI
00147
        /* Initialize MPI... */
00148
        MPI_Init(&argc, &argv);
00149
         MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00150
        MPI_Comm_size(MPI_COMM_WORLD, &size);
00151 #endif
00152
00153
         /* Check arguments... */
00154
        if (argc < 5)
00155
           ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00156
        /* Open directory list... */
if (!(dirlist = fopen(argv[1], "r")))
00157
00158
           ERRMSG("Cannot open directory list!");
00159
00160
00161
         /\star Loop over directories... \star/
         while (fscanf(dirlist, "%s", dirname) != EOF) {
00162
00163
           /* MPI parallelization... */
00164
00165
           if ((++ntask) % size != rank)
00166
             continue;
00167
00168
00169
              Initialize model run...
00170
00171
00172
           /* Set timers... */
00173
           START_TIMER(TIMER_TOTAL);
00174
           START_TIMER (TIMER_INIT);
00175
00176
           /* Allocate... */
00177
           ALLOC(atm, atm_t, 1);
ALLOC(met0, met_t, 1);
ALLOC(met1, met_t, 1);
00178
00179
00180
           ALLOC(dt, double,
00181
                  NP);
00182
           /* Read control parameters... */
sprintf(filename, "%s/%s", dirname, argv[2]);
read_ctl(filename, argc, argv, &ctl);
00183
00184
00185
00186
00187
            /* Initialize random number generators... */
00188
           gsl_rng_env_setup();
           if (omp_get_max_threads() > NTHREADS)
00189
           ERRMSG("Too many threads!");
for (i = 0; i < NTHREADS; i++)
00190
00191
00192
             rng[i] = gsl_rng_alloc(gsl_rng_default);
00193
           /* Read atmospheric data... */
sprintf(filename, "%s/%s", dirname, argv[3]);
read_atm(filename, &ctl, atm);
00194
00195
00196
00197
00198
            /* Get simulation time interval... */
00199
           init_simtime(&ctl, atm);
00200
00201
            /* Get rounded start time... */
00202
           if (ctl.direction == 1)
00203
             t0 = floor(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00204
00205
             t0 = ceil(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00206
00207
            /* Set timers ... */
           STOP_TIMER(TIMER_INIT);
00208
00209
00210
00211
              Loop over timesteps...
00212
00213
00214
           /* Loop over timesteps... */
00215
           for (t = t0; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;</pre>
```

```
t += ctl.direction * ctl.dt_mod) {
00217
00218
             /* Adjust length of final time step... */
00219
             if (ctl.direction * (t - ctl.t_stop) > 0)
00220
              t = ctl.t stop;
00221
             /* Set time steps for air parcels... */
00223
             for (ip = 0; ip < atm\rightarrownp; ip++)
00224
              if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
                    00225
00226
00227
                 dt[ip] = t - atm->time[ip];
00228
              else
00229
                dt[ip] = GSL_NAN;
00230
00231
             /* Get meteorological data... */
00232
             START TIMER (TIMER INPUT):
            get_met(&ctl, argv[4], t, met0, met1);
if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00233
              printf("Warning: Violation of CFL criterion! Set DT_MOD <= %g s!\n",</pre>
00235
00236
                      fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.);
00237
             STOP_TIMER(TIMER_INPUT);
00238
             /* Initialize isosurface... */
00239
00240
             START_TIMER(TIMER_ISOSURF);
00241
            if (t == t0)
00242
              module_isosurf(&ctl, met0, met1, atm, -1);
00243
             STOP_TIMER(TIMER_ISOSURF);
00244
00245
             /* Advection... */
00246
            START_TIMER (TIMER_ADVECT);
00247 #pragma omp parallel for default(shared) private(ip)
00248
            for (ip = 0; ip < atm->np; ip++)
00249
             if (gsl_finite(dt[ip]))
00250
                module_advection(met0, met1, atm, ip, dt[ip]);
00251
            STOP_TIMER(TIMER_ADVECT);
00252
             /* Turbulent diffusion...
00254
             START_TIMER(TIMER_DIFFTURB);
00255 #pragma omp parallel for default(shared) private(ip)
00256
             for (ip = 0; ip < atm->np; ip++)
              if (gsl_finite(dt[ip]))
00257
00258
                module_diffusion_turb(&ctl, atm, ip, dt[ip],
                                        rng[omp_get_thread_num()]);
00259
00260
            STOP_TIMER(TIMER_DIFFTURB);
00261
00262
             /* Mesoscale diffusion... */
00263
            START_TIMER (TIMER_DIFFMESO);
00264 #pragma omp parallel for default(shared) private(ip)
            for (ip = 0; ip < atm->np; ip++)
00265
00266
              if (gsl_finite(dt[ip]))
00267
                module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
00268
                                        rng[omp_get_thread_num()]);
00269
            STOP TIMER (TIMER DIFFMESO);
00270
00271
             /* Sedimentation...
            START_TIMER(TIMER_SEDI);
00273 #pragma omp parallel for default(shared) private(ip)
            for (ip = 0; ip < atm->np; ip++)
00274
00275
              if (gsl_finite(dt[ip]))
00276
                module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
00277
            STOP TIMER (TIMER SEDI);
00278
00279
00280
            START_TIMER(TIMER_ISOSURF);
00281 #pragma omp parallel for default(shared) private(ip)
00282
            for (ip = 0; ip < atm->np; ip++)
  module_isosurf(&ctl, met0, met1, atm, ip);
00283
00284
            STOP TIMER (TIMER ISOSURF);
00285
00286
             /* Position... */
00287
            START_TIMER(TIMER_POSITION);
00288 \#pragma omp parallel for default(shared) private(ip)
            for (ip = 0; ip < atm->np; ip++)
  module_position(met0, met1, atm, ip);
00289
00290
00291
             STOP_TIMER(TIMER_POSITION);
00292
00293
             /* Meteorological data... */
00294
            START_TIMER(TIMER_METEO);
00295 module_meteo(&ctl, met0, met1, atm, 0);
00296 #pragma omp parallel for default(shared) private(ip)
00297
            for (ip = 1; ip < atm->np; ip++)
00298
              module_meteo(&ctl, met0, met1, atm, ip);
00299
            STOP_TIMER(TIMER_METEO);
00300
00301
             /* Decav... */
            START_TIMER(TIMER_DECAY);
00302
```

```
00303 #pragma omp parallel for default(shared) private(ip)
          for (ip = 0; ip < atm->np; ip++)
              if (gsl_finite(dt[ip]))
00305
                 module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00306
00307
             STOP_TIMER(TIMER_DECAY);
00308
00309
              /* Write output...
00310
             START_TIMER(TIMER_OUTPUT);
00311
             write_output(dirname, &ctl, met0, met1, atm, t);
00312
             STOP_TIMER(TIMER_OUTPUT);
00313
           }
00314
00315
00316
              Finalize model run...
00317
00318
00319
           /* Report timers...
           STOP_TIMER(TIMER_TOTAL);
00320
           PRINT_TIMER (TIMER_TOTAL);
00321
           PRINT_TIMER (TIMER_INIT);
00322
00323
           PRINT_TIMER (TIMER_STAGE);
00324
           PRINT_TIMER(TIMER_INPUT);
00325
           PRINT_TIMER (TIMER_OUTPUT);
           PRINT_TIMER (TIMER_ADVECT);
00326
00327
           PRINT_TIMER(TIMER_DECAY);
           PRINT_TIMER (TIMER_DIFFMESO);
00328
00329
           PRINT_TIMER(TIMER_DIFFTURB);
00330
           PRINT_TIMER(TIMER_ISOSURF);
00331
           PRINT_TIMER (TIMER_METEO);
           PRINT_TIMER (TIMER_POSITION);
00332
00333
           PRINT_TIMER (TIMER_SEDI);
00334
00335
           /* Report memory usage... */
           printf("MEMORY_ATM = %g MByte\n", sizeof(atm_t) / 1024. / 1024.);
printf("MEMORY_METEO = %g MByte\n", 2. * sizeof(met_t) / 1024. / 1024.);
printf("MEMORY_DYNAMIC = %g MByte\n",
00336
00337
00338
00339
                   4 * NP * sizeof(double) / 1024. / 1024.);
           printf("MEMORY_STATIC = %g MByte\n",
00340
                   ((3 * GX * GY + 4 * GX * GY * GZ) * sizeof(double)
+ (EX * EY + EX * EY * EP) * sizeof(float)
+ (GX * GY + GX * GY * GZ) * sizeof(int)) / 1024. / 1024.);
00341
00342
00343
00344
           /* Report problem size... */
printf("SIZE_NP = %d\n", atm->np);
00345
00346
00347
           printf("SIZE_TASKS = %d\n", size);
           printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00348
00349
           /* Free random number generators... */
for (i = 0; i < NTHREADS; i++)</pre>
00350
00351
            gsl_rng_free(rng[i]);
00352
00353
00354
           /* Free... */
00355
           free(atm);
00356
           free (met0);
00357
           free (met1);
00358
           free(dt);
00359
00360
00361 #ifdef MPI
00362
        /* Finalize MPI... */
        MPI_Finalize();
00363
00364 #endif
00365
00366
         return EXIT_SUCCESS;
00367 }
00368
00370
00371 void init_simtime(
00372
        ctl_t * ctl,
00373
        atm_t * atm) {
00374
00375
         /* Set inital and final time... */
        if (ctl->direction == 1) {
   if (ctl->t_start < -1e99)</pre>
00376
00377
00378
             ctl->t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00379
           if (ctl->t_stop < -1e99)
        ctl->t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
} else if (ctl->direction == -1) {
00380
00381
          if (ctl->t_stop < -1e99)
00382
           ctl->t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
if (ctl->t_start < -1e99)</pre>
00383
00384
00385
             ctl->t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00386
00387
00388
         /* Check time... */
00389
         if (ctl->direction * (ctl->t stop - ctl->t start) <= 0)
```

```
ERRMSG("Nothing to do!");
00391 }
00392
00394
00395 void module_advection(
      met_t * met0,
00397
       met_t * met1,
00398
       atm_t * atm,
00399
       int ip,
00400
      double dt) {
00401
00402
       double v[3], xm[3];
00403
00404
       /* Interpolate meteorological data... */
       00405
00406
                      &v[0], &v[1], &v[2], NULL, NULL);
00407
00408
00409
       /\star Get position of the mid point... \star/
       00410
00411
       xm[2] = atm -> p[ip] + 0.5 * dt * v[2];
00412
00413
00414
       /* Interpolate meteorological data for mid point... */
       intpol_met_time(met0, met1, atm->time[ip] + 0.5 * dt,
00415
00416
                      xm[2], xm[0], xm[1], NULL, NULL,
00417
                      &v[0], &v[1], &v[2], NULL, NULL);
00418
00419
       /* Save new position... */
00420
      atm->time[ip] += dt;
atm->lon[ip] += dx2deg(dt * v[0] / 1000., xm[1]);
atm->lat[ip] += dy2deg(dt * v[1] / 1000.);
00421
00422
00423
       atm->p[ip] += dt * v[2];
00424 }
00425
00427
00428 void module_decay(
      ctl_t * ctl,
met_t * met0,
00429
00430
00431
       met_t * met1,
       atm_t * atm,
00432
00433
       int ip,
00434
      double dt) {
00435
00436
      double ps, pt, tdec;
00437
00438
      /* Check lifetime values... */
       if ((ctl->tdec_trop <= 0 && ctl->tdec_strat <= 0) || ctl->
00439
     qnt_m < 0
00440
00441
00442
       /* Set constant lifetime... */
      if (ctl->tdec_trop == ctl->tdec_strat)
  tdec = ctl->tdec_trop;
00443
00444
00445
00446
      /* Set altitude-dependent lifetime... */
00447
       else {
00448
        /* Get surface pressure... */
intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00449
00450
                       atm->lon[ip], atm->lat[ip], &ps, NULL,
00451
00452
                       NULL, NULL, NULL, NULL, NULL);
00453
00454
         /* Get tropopause pressure... */
00455
        pt = tropopause(atm->time[ip], atm->lat[ip]);
00456
00457
         /* Set lifetime... */
         if (atm->p[ip] <= pt)</pre>
00458
00459
          tdec = ctl->tdec_strat;
00460
         else
00461
          tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
     p[ip]);
00462
00463
00464
       /* Calculate exponential decay... */
00465
       atm->q[ctl->qnt_m][ip] *= exp(-dt / tdec);
00466 }
00467
00469
00470 void module_diffusion_meso(
00471
      ctl_t * ctl,
00472
       met_t * met0,
       met_t * met1,
00473
      atm_t * atm,
00474
```

```
00475
         int ip,
00476
         double dt,
00477
         gsl_rng * rng) {
00478
00479
         double r, rs, u[16], v[16], w[16], usig, vsig, wsig;
00480
         int ix, iy, iz;
00481
00482
00483
          /* Calculate mesoscale velocity fluctuations... */
00484
         if (ctl->turb_meso > 0) {
00485
00486
            /* Get indices... */
00487
            ix = locate(met0->lon, met0->nx, atm->lon[ip]);
00488
            iy = locate(met0->lat, met0->ny, atm->lat[ip]);
00489
            iz = locate(met0->p, met0->np, atm->p[ip]);
00490
00491
            /* Collect local wind data... */
            u[0] = met0->u[ix][iy][iz];
u[1] = met0->u[ix + 1][iy][iz];
00492
00493
00494
            u[2] = met0 -> u[ix][iy + 1][iz];
            u[3] = met0->u[ix + 1][iy + 1][iz];
u[4] = met0->u[ix][iy][iz + 1];
00495
00496
             u[5] = met0 -> u[ix + 1][iy][iz + 1]; 
 u[6] = met0 -> u[ix][iy + 1][iz + 1]; 
 u[7] = met0 -> u[ix + 1][iy + 1][iz + 1]; 
00497
00498
00499
00500
00501
            v[0] = met0 -> v[ix][iy][iz];
00502
            v[1] = met0 -> v[ix + 1][iy][iz];
00503
            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];
00504
00505
00506
            v[5] = met0 -> v[ix + 1][iy][iz + 1];
00507
            v[6] = met0 -> v[ix][iy + 1][iz + 1];
00508
            v[7] = met0 -> v[ix + 1][iy + 1][iz + 1];
00509
            w[0] = met0->w[ix][iy][iz];
00510
            w[1] = met0->w[ix + 1][iy][iz];
w[2] = met0->w[ix][iy + 1][iz];
00511
            w[3] = met0->w[ix + 1][iy + 1][iz];
w[4] = met0->w[ix][iy][iz + 1];
00513
00514
00515
            w[5] = met0 -> w[ix + 1][iy][iz + 1];
             w[6] = met0 -> w[ix][iy + 1][iz + 1]; 
 w[7] = met0 -> w[ix + 1][iy + 1][iz + 1]; 
00516
00517
00518
00519
             /* Get indices... */
00520
            ix = locate(met1->lon, met1->nx, atm->lon[ip]);
00521
            iy = locate(met1->lat, met1->ny, atm->lat[ip]);
00522
            iz = locate(met1->p, met1->np, atm->p[ip]);
00523
00524
            /* Collect local wind data... */
00525
            u[8] = met1->u[ix][iy][iz];
00526
            u[9] = met1->u[ix + 1][iy][iz];
00527
            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];
00528
00529
            u[13] = met1->u[ix + 1][iy][iz + 1];
u[14] = met1->u[ix][iy + 1][iz + 1];
00530
00531
00532
            u[15] = met1 -> u[ix + 1][iy + 1][iz + 1];
00533
00534
            v[8] = met1->v[ix][iy][iz];
            v[9] = met1->v[ix + 1][iy][iz];
v[10] = met1->v[ix][iy + 1][iz];
00535
00536
            v[11] = met1->v[ix + 1][iy + 1][iz];
v[12] = met1->v[ix][iy][iz + 1];
00537
00538
00539
            v[13] = met1 -> v[ix + 1][iy][iz + 1];
00540
            v[14] = met1->v[ix][iy + 1][iz + 1];
            v[15] = met1 -> v[ix + 1][iy + 1][iz + 1];
00541
00542
00543
            w[8] = met1->w[ix][iy][iz];
            w[9] = met1->w[ix + 1][iy][iz];
00544
00545
            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];
00546
00547
            w[13] = met1->w[ix + 1][iy][iz + 1];
w[14] = met1->w[ix][iy + 1][iz + 1];
w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00548
00549
00550
00551
00552
            /* Get standard deviations of local wind data... */
00553
            usig = gsl_stats_sd(u, 1, 16);
            vsig = gsl_stats_sd(v, 1, 16);
wsig = gsl_stats_sd(v, 1, 16);
wsig = gsl_stats_sd(w, 1, 16);
00554
00555
00557
             /\star Set temporal correlations for mesoscale fluctuations... \star/
00558
            r = 1 - 2 * fabs(dt) / ctl->dt_met;
00559
            rs = sqrt(1 - r * r);
00560
00561
            /* Calculate mesoscale wind fluctuations... */
```

```
atm->up[ip] = (float)
00563
           (r * atm->up[ip]
00564
             + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_meso * usig));
00565
          atm->vp[ip] = (float)
00566
           (r * atm->vp[ip]
          + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_meso * vsig));
atm->wp[ip] = (float)
00567
00568
00569
            (r * atm->wp[ip]
00570
             + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_meso * wsig));
00571
          /* 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.);
00572
00573
00574
00575
          atm->p[ip] += atm->wp[ip] * dt;
00576
00577 }
00578
00579 /
       00580
00581 void module_diffusion_turb(
       ctl_t * ctl,
atm_t * atm,
00582
00583
00584
        int ip,
00585
        double dt.
00586
       gsl_rng * rng) {
00587
00588
        double dx, dz, pt, p0, p1, w;
00589
00590
       /* Get tropopause pressure... */
00591
       pt = tropopause(atm->time[ip], atm->lat[ip]);
00592
00593
        /* Get weighting factor... */
        p1 = pt * 0.866877899;
p0 = pt / 0.866877899;
00594
00595
00596
        if (atm->p[ip] > p0)
        w = 1;
else if (atm->p[ip] < p1)</pre>
00597
00598
00599
00600
        else
00601
          w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00602
00603
        /* Set diffusivitiy... */
       dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00604
00605
        dz = w * ctl - turb_dz_trop + (1 - w) * ctl - turb_dz_strat;
00606
00607
        /* Horizontal turbulent diffusion... */
00608
        if (dx > 0) {
00609
         atm->lon[ip]
00610
            += dx2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00611
                      / 1000., atm->lat[ip]);
00612
          atm->lat[ip]
00613
            += dy2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00614
                      / 1000.);
00615
00616
00617
        /* Vertical turbulent diffusion... */
        if (dz > 0)
00618
00619
         atm->p[ip]
00620
            += dz2dp(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dz * fabs(dt)))
00621
                     / 1000., atm->p[ip]);
00622 }
00623
00625
00626 void module_isosurf(
00627
       ctl_t * ctl,
        met_t * met0,
00628
00629
        met_t * met1,
       atm_t * atm,
00630
00631
        int ip) {
00632
00633
       static double *iso, *ps, t, *ts;
00634
00635
       static int idx, ip2, n;
00636
00637
       FILE *in:
00638
00639
        char line[LEN];
00640
        /* Check control parameter... */
if (ctl->isosurf < 1 || ctl->isosurf > 4)
00641
00642
00643
          return;
00644
00645
        /* Initialize... */
00646
        if (ip < 0) {</pre>
00647
00648
          /* Allocate... */
```

```
ALLOC(iso, double,
                 NP);
00650
00651
           ALLOC(ps, double,
00652
                 NP);
00653
           ALLOC(ts, double,
00654
                 NP);
00656
           /* Save pressure... */
00657
           if (ctl->isosurf == 1)
00658
             for (ip2 = 0; ip2 < atm->np; ip2++)
               iso[ip2] = atm->p[ip2];
00659
00660
           /* Save density... */
else if (ctl->isosurf == 2)
00661
00662
00663
             for (ip2 = 0; ip2 < atm->np; ip2++) {
               00664
00665
00666
               iso[ip2] = atm->p[ip2] / t;
00667
00668
00669
00670
           /\star Save potential temperature... \star/
           else if (ct1->isosurf == 3)
   for (ip2 = 0; ip2 < atm->np; ip2++) {
00671
00672
00673
               intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
                                 atm->lon[ip2], atm->lat[ip2], NULL, &t, NULL, NULL,
00674
00675
                                 NULL, NULL, NULL);
00676
               iso[ip2] = t * pow(P0 / atm->p[ip2], 0.286);
00677
00678
00679
           /* Read balloon pressure data... */
00680
           else if (ctl->isosurf == 4) {
00681
00682
             /\star Write info... \star/
00683
             printf("Read balloon pressure data: %s\n", ctl->balloon);
00684
             /* Open file... */
if (!(in = fopen(ctl->balloon, "r")))
00685
00687
               ERRMSG("Cannot open file!");
00688
00689
             /\star Read pressure time series... \star/
             while (fgets(line, LEN, in))
  if (sscanf(line, "%lg %lg", &ts[n], &ps[n]) == 2)
00690
00691
                  if ((++n) > NP)
00692
                   ERRMSG("Too many data points!");
00693
00694
00695
             /* Check number of points... */
00696
             if (n < 1)
               ERRMSG("Could not read any data!");
00697
00698
00699
             /* Close file... */
00700
             fclose(in);
00701
00702
00703
           /* Leave initialization... */
00704
           return;
00705
00706
00707
         /* Restore pressure... */
00708
        if (ctl->isosurf == 1)
00709
          atm->p[ip] = iso[ip];
00710
00711
        /* Restore density... */
else if (ctl->isosurf == 2) {
00712
00713
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00714
                             atm->lat[ip], NULL, &t, NULL, NULL, NULL, NULL, NULL);
          atm->p[ip] = iso[ip] * t;
00715
00716
00717
00718
         /* Restore potential temperature... */
00719
        else if (ctl->isosurf == 3) {
00720
          intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
          \label{eq:atm-power} $$ atm->lat[ip], NULL, &t, NULL, NULL, NULL, NULL, NULL); $$ atm->p[ip] = P0 * pow(iso[ip] / t, -1. / 0.286);
00721
00722
00723
00724
        /* Interpolate pressure... */
else if (ctl->isosurf == 4) {
00725
00726
         if (atm->time[ip] <= ts[0])</pre>
00727
           atm->p[ip] = ps[0];
else if (atm->time[ip] >= ts[n - 1])
00729
00730
            atm->p[ip] = ps[n-1];
00731
           else {
            idx = locate(ts, n, atm->time[ip]);
atm->p[ip] = LIN(ts[idx], ps[idx],
00732
00733
```

```
ts[idx + 1], ps[idx + 1], atm->time[ip]);
00735
00736
00737 }
00738
00739 /
        ********************************
00741 void module_meteo(
        ctl_t * ctl,
met_t * met0,
00742
00743
00744
        met_t * met1,
00745
        atm t * atm.
00746
        int ip) {
00747
00748
        static FILE *in;
00749
00750
        static char filename[LEN], line[LEN];
00751
00752
        static double lon[GX], lat[GY], var[GX][GY],
00753
          rdum, rlat, rlat_old = -999, rlon, rvar;
00754
00755
        static int year_old, mon_old, day_old, nlon, nlat;
00756
        double a, b, c, ps, p1, p_hno3, p_h2o, t, t1, u, u1, v, v1, w,
    x1, x2, h2o, o3, grad, vort, var0, var1;
00757
00758
00759
00760
        int day, mon, year, idum, ilat, ilon;
00761
00762
        /* Interpolate meteorological data... */
00763
        intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00764
                           atm->lat[ip], &ps, &t, &u, &v, &w, &h2o, &o3);
00765
00766
         /\star Set surface pressure... \star/
00767
        if (ctl->qnt_ps >= 0)
00768
          atm->q[ctl->qnt_ps][ip] = ps;
00769
00770
        /* Set pressure... */
00771
        if (ctl->qnt_p >= 0)
00772
          atm->q[ctl->qnt_p][ip] = atm->p[ip];
00773
00774
        /* Set temperature... */
00775
        if (ctl->qnt_t >= 0)
          atm->q[ctl->qnt_t][ip] = t;
00776
00777
00778
        /* Set zonal wind... */
00779
        if (ctl->qnt_u >= 0)
00780
          atm->q[ctl->qnt_u][ip] = u;
00781
00782
        /* Set meridional wind... */
00783
        if (ctl->qnt_v >= 0)
00784
           atm \rightarrow q[ctl \rightarrow qnt_v][ip] = v;
00785
00786
         /* Set vertical velocity... */
00787
        if (ctl->qnt_w >= 0)
00788
          atm->q[ctl->qnt_w][ip] = w;
00789
00790
         /* Set water vapor vmr... */
00791
        if (ctl->qnt_h2o >= 0)
00792
           atm->q[ctl->qnt_h2o][ip] = h2o;
00793
00794
        /* Set ozone vmr... */
if (ctl->qnt_o3 >= 0)
00795
00796
          atm->q[ctl->qnt_o3][ip] = o3;
00797
00798
         /* Calculate potential temperature... */
         if (ctl->qnt_theta >= 0)
00799
00800
          atm \rightarrow q[ctl \rightarrow qnt\_theta][ip] = t * pow(P0 / atm \rightarrow p[ip], 0.286);
00801
00802
         /* Calculate potential vorticity... */
00803
         if (ctl->qnt_pv >= 0) {
00804
           /* Absolute vorticity... */
vort = 2 * 7.2921e-5 * sin(atm->lat[ip] * M_PI / 180.);
if (fabs(atm->lat[ip]) < 89.) {
00805
00806
00807
             intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00808
00809
                               (atm->lon[ip] >=
00810
                                0 ? atm->lon[ip] - 1. : atm->lon[ip] + 1.),
             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]));
00811
00812
00813
00814
           intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00816
                             (atm->lat[ip] >=
                              0 ? atm->lat[ip] - 1. : atm->lat[ip] + 1.), NULL, NULL,
00817
00818
                             &ul, NULL, NULL, NULL, NULL);
```

```
vort += (ul - u) / 1000. / ((atm->lat[ip] >= 0 ? -1 : 1) * deg2dy(1.));
00820
00821
          /* Potential temperature gradient... */
00822
          p1 = 0.85 * atm->p[ip];
          00823
00824
00826
            / (100. * (p1 - atm->p[ip]));
00827
           /* Calculate PV... */
00828
          atm->q[ctl->qnt_pv][ip] = -1e6 * GO * vort * grad;
00829
00830
00831
00832
        /* Calculate T_ice (Marti and Mauersberger, 1993)... */
00833
        if (ctl->qnt_tice >= 0 || ctl->qnt_tsts >= 0)
00834
          atm->q[ctl->qnt_tice][ip] =
00835
             -2663.5 /
00836
             (\log 10((ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] * 100.) -
00837
00838
00839
         /* Calculate T_NAT (Hanson and Mauersberger, 1988)... \star/
        if (ctl->qnt_tnat >= 0 || ctl->qnt_tsts >= 0) {
   if (ctl->psc_hno3 > 0)
00840
00841
00842
            p_hno3 = ct1->psc_hno3 * atm->p[ip] / 1.333224;
00843
            p_hno3 = module_meteo_hno3(atm->time[ip], atm->lat[ip], atm->
      p[ip])
00845
               * 1e-9 * atm->p[ip] / 1.333224;
          p_h2o = (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
a = 0.009179 - 0.00088 * log10(p_h2o);
00846
00847
          b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
00848
00849
          c = -11397.0 / a;
00850
          x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
00851
          x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
00852
          if (x1 > 0)
00853
            atm->q[ctl->qnt_tnat][ip] = x1;
00854
          if (x2 > 0)
            atm->q[ctl->qnt_tnat][ip] = x2;
00856
00857
00858
        /* Calculate T_STS (mean of T_ice and T_NAT)... \star/
00859
        if (ctl->qnt_tsts >= 0) {
         if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)</pre>
00860
00861
            ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
          atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
00862
00863
                                                + atm->q[ctl->qnt_tnat][ip]);
00864
00865
00866
        /* Read variance data for current day... */
        if (ip == 0 && ctl->qnt_gw_var >= 0) {
   jsec2time(atm->time[ip], &year, &mon, &day, &idum, &idum, &idum, &rdum);
00867
00868
00869
           if (year != year_old || mon != mon_old || day != day_old) {
             year_old = year;
00870
             mon_old = mon;
00871
             day_old = day;
00872
00873
            nlon = nlat = -1;
            sprintf(filename, "%s_%d_%02d_%02d.tab",
            ctl->gw_basename, year, mon, day);
if ((in = fopen(filename, "r"))) {
00875
00876
              printf("Read gravity wave data: %s\n", filename);
00877
               while (fgets(line, LEN, in)) {
  if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rvar) != 3)
00878
00879
00880
                   continue;
                 if (rlat != rlat_old) {
00881
00882
                   rlat_old = rlat;
00883
                   if ((++nlat) > GY)
00884
                     ERRMSG("Too many latitudes!");
00885
                   nlon = -1;
00886
00887
                 if ((++nlon) > GX)
00888
                   ERRMSG("Too many longitudes!");
                 lon[nlon] = rlon;
lat[nlat] = rlat;
00889
00890
                 var[nlon][nlat] = GSL_MAX(0, rvar);
00891
00892
00893
               fclose(in);
00894
               nlat++;
00895
              nlon++;
00896
               printf("Warning: Missing gravity wave data: %s\n", filename);
00897
00898
          }
00899
        }
00900
00901
        /* Interpolate variance data... */
        if (ctl->qnt_gw_var >= 0) {
  if (nlat >= 2 && nlon >= 2) {
00902
00903
            ilat = locate(lat, nlat, atm->lat[ip]);
00904
```

```
ilon = locate(lon, nlon, atm->lon[ip]);
                var0 = LIN(lat[ilat], var[ilon][ilat],
    lat[ilat + 1], var[ilon][ilat + 1], atm->lat[ip]);
00906
00907
                00908
00909
                atm->q[ctl->qnt_qw_var][ip]
00910
00911
                 = LIN(lon[ilon], var0, lon[ilon + 1], var1, atm->lon[ip]);
00912
00913
                atm->q[ctl->qnt_gw_var][ip] = GSL_NAN;
00914
00915 }
00916
00917 /
00918
00919 double module_meteo_hno3(
          double t,
00920
00921
          double lat.
00922
          double p) {
00923
00924
          static double secs[12] = { 1209600.00, 3888000.00, 6393600.00,
00925
              9072000.00, 11664000.00, 14342400.00,
00926
             16934400.00, 19612800.00, 22291200.00,
00927
             24883200.00, 27561600.00, 30153600.00
00928
00929
00930
           static double lats[18] = { -85, -75, -65, -55, -45, -35, -25, -15, -5,
00931
             5, 15, 25, 35, 45, 55, 65, 75, 85
00932
00933
          static double ps[10] = { 4.64159, 6.81292, 10, 14.678, 21.5443,
00934
00935
             31.6228, 46.4159, 68.1292, 100, 146.78
00936
00937
00938
           static double hno3[12][18][10] = {
             {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74}, {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57}, {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
00939
00940
00941
               {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00943
               \{0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709\},\
               {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37}, {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244}, {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222}, {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181}, {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00944
00945
00946
00947
00948
               {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985}, {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00950
00951
               {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
               {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77}, {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49}, {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00952
00953
00954
               {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
               {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}}
00956
00957
              {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
               {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42}, {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
00958
00959
               {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00960
               {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661}
               \{0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332\},
00962
00963
               {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
00964
               {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
               {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167}, {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101}, {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00965
00966
               {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191}, {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
00968
00969
00970
               {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
00971
               {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01}, {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
00972
               {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
00973
                {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
00975
              {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00976
               {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
               {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3}, {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98}, {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642}, {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33}
00977
00978
00979
00980
               {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
00981
00982
               {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
00983
               {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
00984
               \{0.945,\ 1.69,\ 2.27,\ 2.64,\ 2.83,\ 2.2,\ 1.83,\ 0.561,\ 0.139,\ 0.121\},
00985
               \{0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135\},
               {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
               \{0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1\},\
00987
               {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12}, {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82}, {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54}, {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
00988
00989
00990
00991
```

```
{1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
              {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75}, {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00993
00994
00995
               {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
00996
               {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09}, {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
00997
               {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
                {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
00999
01000
                {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12}
01001
                \{0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172\},
               {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
01002
               {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
01003
01004
               \{0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286\},\
                {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
01005
01006
                {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
01007
                {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
01008
               {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
01009
               {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
               {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
              {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
01011
               {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57}, {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63}, {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37}, {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88}, {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
01012
01013
01014
01015
01016
               {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
                {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
01018
01019
                {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
               {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183}, {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18}, {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343}, {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
01020
01021
01022
             {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964}, {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83}, {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25}, {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39}, {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52}, {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}}, {1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
01024
01025
01026
01027
01028
01030
               \{1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05\},\
                {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65}
01031
01032
                {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
               {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
01033
               {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
01034
01035
               {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
01036
               {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
01037
                {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
               {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194}, {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229}, {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
01038
01039
01040
               {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
01041
               \{0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41\},
                {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
01043
01044
                {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9}
               {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88}, {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}},
01045
01046
              {3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2. {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
01047
               {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
01049
               {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3}, {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
01050
01051
               {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656}, {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
01052
01053
               {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
               {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12}
{1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
01055
01056
01057
                {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
01058
               \{0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259\},
               \{0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422\},
01059
01060
               \{0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913\},
                {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
                {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
01062
01063
                \{0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61\},
              {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}, {5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4}, {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73}, {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
01064
01065
01066
01067
                {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14}
01068
               {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38}, {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
01069
01070
                {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
01071
01072
               \{0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181\},
                {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
                {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
01074
01075
                {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
01076
                {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
               {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341}, {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
01077
01078
```

```
{0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
              {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
01080
01081
              {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5}
              {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
01082
             {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
01083
              {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
01084
              \{0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09\},\
              {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
01086
              {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22}, {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646}, {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
01087
01088
01089
              {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
01090
              \{0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815\},
01091
              {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
01092
01093
              {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
              {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163}, {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
01094
01095
              {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714}, {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
01096
              \{0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41\},
01098
              {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
01099
01100
              {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1,
                                                                          3.43, 1.65}}
01101
            \{\{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91\},
              {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84}, {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
01102
01103
              {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56, 
{0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
01104
01105
01106
              {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
01107
              \{0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21\}
01108
              {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
              {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
01109
              \{0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146\},
01110
              {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
01111
01112
              {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
              {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353}, {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
01113
01114
              {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2}, {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
01115
              {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
              {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
01118
01119
            {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78}
              {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73}, {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
01120
01121
              {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
01122
              \{0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955\},
01124
              {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61}
01125
              {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
01126
              {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
              \{0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121\},
01127
              {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147}, {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
01128
              {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172}, {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
01130
01131
              {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948}, {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
01132
01133
              {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
01134
              {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
              {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
01136
            {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89}}
{0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
01137
01138
01139
              \{0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65\},
              {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28}, {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837}
01140
01141
                                                                                  0.837}
              {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
              {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
01143
01144
              {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
01145
              {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173}, {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
01146
              {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
01147
              {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
              {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
01149
01150
              {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
01151
              {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
              {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24}, {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35}, {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
01152
01153
01154
01155
01156
01157
         double aux00, aux01, aux10, aux11, sec;
01158
01159
         int ilat, ip, isec;
01160
01161
          /* Get seconds since begin of year... */
01162
          sec = fmod(t, 365.25 * 86400.);
01163
01164
          /* Get indices... */
          ilat = locate(lats, 18, lat);
01165
```

```
01166
        ip = locate(ps, 10, p);
        isec = locate(secs, 12, sec);
01167
01168
01169
        /* Interpolate... */
        01170
01171
        aux01 = LIN(ps[ip], hno3[isec][ilat + 1][ip],
01172
01173
                     ps[ip + 1], hno3[isec][ilat + 1][ip + 1], p);
01174
        aux10 = LIN(ps[ip], hno3[isec + 1][ilat][ip],
        ps[ip + 1], hno3[isec + 1][ilat][ip + 1], p);

aux11 = LIN(ps[ip], hno3[isec + 1][ilat + 1][ip],

ps[ip + 1], hno3[isec + 1][ilat + 1][ip + 1], p);

aux00 = LIN(lats[ilat], aux00, lats[ilat + 1], aux01, lat);

aux11 = LIN(lats[ilat], aux10, lats[ilat + 1], aux11, lat);
01175
01176
01177
01178
01179
01180
        return LIN(secs[isec], aux00, secs[isec + 1], aux11, sec);
01181 }
01182
        01183 /
01184
01185 void module_position(
01186
        met_t * met0,
01187
        met_t * met1,
        atm t * atm,
01188
01189
        int ip) {
01190
01191
        double ps;
01192
01193
         /* Calculate modulo... */
        atm->lon[ip] = fmod(atm->lon[ip], 360);
atm->lat[ip] = fmod(atm->lat[ip], 360);
01194
01195
01196
01197
         /* Check latitude... */
        while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
  if (atm->lat[ip] > 90) {
01198
01199
            atm->lat[ip] = 180 - atm->lat[ip];
atm->lon[ip] += 180;
01200
01201
01202
          if (atm->lat[ip] < -90) {</pre>
            atm->lat[ip] = -180 - atm->lat[ip];
atm->lon[ip] += 180;
01204
01205
01206
        }
01207
01208
01209
        /* Check longitude... */
        while (atm->lon[ip] < -180)
01210
01211
          atm->lon[ip] += 360;
01212
        while (atm->lon[ip] >= 180)
01213
          atm->lon[ip] -= 360;
01214
01215
        /* Get surface pressure... */
intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
01216
01217
                         atm->lon[ip], atm->lat[ip], &ps, NULL,
01218
                          NULL, NULL, NULL, NULL, NULL);
01219
01220
        /* Check pressure... */
        if (atm->p[ip] > ps)
01221
          atm->p[ip] = ps;
01223
        else if (atm->p[ip] < met0->p[met0->np - 1])
01224
          atm - p[ip] = met0 - p[met0 - np - 1];
01225 }
01226
01228
01229 void module sedi(
01230
        ctl_t * ctl,
01231
        met_t * met0,
01232
        met_t * met1,
atm_t * atm,
01233
01234
        int ip.
        double dt) {
01236
01237
        /\star Coefficients for Cunningham slip-flow correction (Kasten, 1968): \star/
01238
        const double A = 1.249, B = 0.42, C = 0.87;
01239
01240
        /* Specific gas constant for dry air [J/(kg K)]: */
        const double R = 287.058;
01241
01242
01243
        /\star Average mass of an air molecule [kg/molec]: \star/
01244
        const double m = 4.8096e-26;
01245
01246
        double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
01247
01248
         /* Check if parameters are available... */
01249
        if (ctl->qnt_r < 0 || ctl->qnt_rho < 0)</pre>
01250
         return;
01251
01252
        /* Convert units... */
```

```
p = 100 * atm -> p[ip];
        r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
01254
01255
        rho_p = atm->q[ctl->qnt_rho][ip];
01256
01257
       /* Get temperature... */
       intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
01258
     lon[ip],
01259
                        atm->lat[ip], NULL, &T, NULL, NULL, NULL, NULL, NULL);
01260
01261
        /\star Density of dry air... \star/
       rho = p / (R * T);
01262
01263
01264
       /* Dynamic viscosity of air... */
       eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
01265
01266
       /* Thermal velocity of an air molecule... */ v = sqrt(8 * GSL_CONST_MKSA_BOLTZMANN * T / (M_PI * m));
01267
01268
01269
01270
       /* Mean free path of an air molecule... */
01271
       lambda = 2 * eta / (rho * v);
01272
01273
        /* Knudsen number for air... */
01274
       K = lambda / r_p;
01275
01276
        /* Cunningham slip-flow correction... */
01277
        G = 1 + K * (A + B * exp(-C / K));
01278
01279
        /* Sedimentation (fall) velocity... */
01280
01281
          2. * gsl_pow_2(r_p) * (rho_p -
01282
                                 rho) * GSL CONST MKSA GRAV ACCEL / (9. * eta) * G;
01283
01284
        /* Calculate pressure change... */
01285
        atm > p[ip] += dz2dp(v_p * dt / 1000., atm > p[ip]);
01286 }
01287
01290 void write_output(
01291
       const char *dirname,
01292
        ctl_t * ctl,
        met_t * met0,
01293
       met_t * met1,
atm_t * atm,
01294
01295
01296
       double t) {
01297
01298
       char filename[LEN];
01299
01300
       double r:
01301
01302
       int year, mon, day, hour, min, sec;
01303
01304
        /* Get time... */
01305
        jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01306
01307
       /* Write atmospheric data... */
if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01308
01309
         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01310
                 dirname, ctl->atm_basename, year, mon, day, hour, min);
01311
         write_atm(filename, ctl, atm, t);
01312
01313
01314
        /* Write CSI data... */
01315
        if (ctl->csi_basename[0] != '-') {
01316
          sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01317
         write_csi(filename, ctl, atm, t);
01318
01319
01320
        /* Write ensemble data... */
        if (ctl->ens_basename[0] != '-') {
01322
        sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01323
         write_ens(filename, ctl, atm, t);
01324
01325
01326
        /* Write gridded data...
01327
        if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01328
        sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01329
                  dirname, ctl->grid_basename, year, mon, day, hour, min);
01330
         write_grid(filename, ctl, met0, met1, atm, t);
01331
01332
01333
        /* Write profile data...
01334
        if (ctl->prof_basename[0] != '-') {
         sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01335
01336
         write_prof(filename, ctl, met0, met1, atm, t);
01337
01338
```

```
01339  /* Write station data... */
01340  if (ctl->stat_basename[0] != '-') {
01341      sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01342      write_station(filename, ctl, atm, t);
01343  }
01344 }
```

#### 5.35 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[])

#### 5.35.1 Detailed Description

Create meteorological data files with synthetic wind fields.

Definition in file wind.c.

#### 5.35.2 Function Documentation

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

Definition at line 188 of file wind.c.

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

Definition at line 41 of file wind.c.

```
00043
00044
00045
       ctl_t ctl;
00046
00047
       static char filename[LEN];
00048
00049
       static double r, t0, z0, z1, dataLon[EX], dataLat[EY], dataZ[EP],
00050
         u0, u1, alpha;
00051
00052
       static float *dataT, *dataU, *dataV, *dataW;
00053
00054
       static int ncid, dims[4], timid, levid, latid, lonid, tid, uid, vid, wid,
00055
         idx, ix, iy, iz, nx, ny, nz, year, mon, day, hour, min, sec;
00056
        /* Allocate... */
00057
00058
       ALLOC(dataT, float,
EP * EY * EX);
00059
00060
       ALLOC(dataU, float,
00061
              EP * EY * EX);
```

```
ALLOC(dataV, float,
                         EP * EY * EX);
00063
00064
             ALLOC (dataW, float,
                       EP * EY * EX);
00065
00066
00067
              /* Check arguments... */
             if (argc < 3)
00069
                 ERRMSG("Give parameters: <ctl> <metbase>");
00070
00071
             /* Read control parameters... */
00072
             read_ctl(argv[1], argc, argv, &ctl);
              t0 = scan_ctl(argv[1], argc, argv, "WIND_TO", -1, "0", NULL);

nv = (int) ccan_ctl(argv[1], argc, argv, "WIND_TO", -1, "0", NULL);
00073
             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);
00074
00075
             ny = (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);
u0 = scan_ctl(argv[1], argc, argv, "WIND_U0", -1, "38.587660177302", NULL);
u1 = scan_ctl(argv[1], argc, argv, "WIND_U1", -1, "38.587660177302", NULL);
00076
00077
00078
00079
08000
              alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);
00081
00082
00083
              /* Check dimensions... */
00084
             if (nx < 1 || nx > EX)
                ERRMSG("Set 1 <= NX <= MAX!");
00085
00086
              if (ny < 1 || ny > EY)
                ERRMSG("Set 1 <= NY <= MAX!");</pre>
00087
88000
             if (nz < 1 || nz > EP)
00089
                ERRMSG("Set 1 <= NZ <= MAX!");</pre>
00090
00091
             /* Get time... */
00092
             jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
t0 = year * 10000. + mon * 100. + day + hour / 24.;
00093
00094
             /* Set filename... */
sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", argv[2], year, mon, day, hour);
00095
00096
00097
00098
              /* Create netCDF file... */
              NC(nc_create(filename, NC_CLOBBER, &ncid));
00100
             /* 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]));
00101
00102
00103
00104
00105
00106
00107
               /* 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));
00108
00109
00110
00111
              NC(nc_def_var(ncid, "T", NC_FLOAT, 4, &dims[0], &tid));
00112
              NC(nc_def_var(ncid, "U", NC_FLOAT, 4, &dims[0], &uid));
NC(nc_def_var(ncid, "V", NC_FLOAT, 4, &dims[0], &vid));
00113
00114
              NC(nc_def_var(ncid, "W", NC_FLOAT, 4, &dims[0], &wid));
00115
00116
            /* Set attributes... */
add_text_attribute(ncid, "time", "long_name", "time");
add_text_attribute(ncid, "time", "units", "day as %Y$m%d.%f");
add_text_attribute(ncid, "lon", "long_name", "longitude");
add_text_attribute(ncid, "lon", "units", "degrees_east");
add_text_attribute(ncid, "lat", "long_name", "latitude");
add_text_attribute(ncid, "lev", "long_name", "air_pressure");
add_text_attribute(ncid, "lev", "units", "Pa");
add_text_attribute(ncid, "lev", "units", "Pa");
add_text_attribute(ncid, "T", "long_name", "Temperature");
add_text_attribute(ncid, "T", "units", "K");
add_text_attribute(ncid, "U", "long_name", "U velocity");
add_text_attribute(ncid, "U", "units", "m s**-1");
add_text_attribute(ncid, "V", "units", "m s**-1");
add_text_attribute(ncid, "V", "units", "m s**-1");
00117
              /* Set attributes... */
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130
              add_text_attribute(ncid, "V", "units", "m s**-1");
add_text_attribute(ncid, "W", "long_name", "Vertical velocity");
add_text_attribute(ncid, "W", "units", "Pa s**-1");
00131
00132
00133
00134
00135
              /* End definition... */
00136
              NC(nc enddef(ncid));
00137
00138
              /* Set coordinates... */
00139
              for (ix = 0; ix < nx; ix++)
                dataLon[ix] = 360.0 / nx * (double) ix;
00140
             for (iy = 0; iy < ny; iy++)
  dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;</pre>
00141
00142
00143
              for (iz = 0; iz < nz; iz++)
                 dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));
00144
00145
              /* Write coordinates... */
00146
              NC(nc_put_var_double(ncid, timid, &t0));
00147
00148
             NC(nc_put_var_double(ncid, levid, dataZ));
```

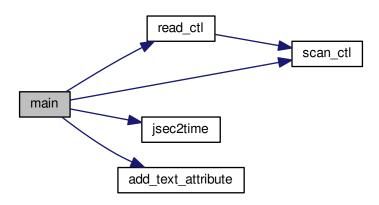
```
NC(nc_put_var_double(ncid, lonid, dataLon));
00150
        NC(nc_put_var_double(ncid, latid, dataLat));
00151
         /* Create wind fields (Williamson et al., 1992)... \star/
00152
        for (ix = 0; ix < nx; ix++)
for (iy = 0; iy < ny; iy++)
00153
00154
00155
             for (iz = 0; iz < nz; iz++) {
00156
               idx = (iz * ny + iy) * nx + ix;
               dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
00157
                                       * (cos(dataLat[iy] * M_PI / 180.0)

* cos(alpha * M_PI / 180.0)

+ sin(dataLat[iy] * M_PI / 180.0)

* cos(dataLon[ix] * M_PI / 180.0)
00158
00159
00160
00161
00162
                                           * sin(alpha * M_PI / 180.0)));
               00163
00164
00165
00166
00167
00168
         /* Write wind data... */
00169
        NC(nc_put_var_float(ncid, tid, dataT));
00170
        NC(nc_put_var_float(ncid, uid, dataU));
00171
        NC(nc_put_var_float(ncid, vid, dataV));
00172
        NC(nc_put_var_float(ncid, wid, dataW));
00173
00174
         /* Close file... */
00175
        NC(nc_close(ncid));
00176
00177
        /* Free... */
        free(dataT);
00178
00179
        free(dataU);
00180
        free (dataV);
00181
        free(dataW);
00182
00183
        return EXIT_SUCCESS;
00184 }
```

Here is the call graph for this function:



#### 5.36 wind.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
```

5.36 wind.c 209

```
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(
00032
         int ncid.
00033
         char *varname,
char *attrname,
00034
00035
         char *text);
00036
00037 /* -----
00038
          Main...
00039
00040
00041 int main(
00042
00043
         char *argv[]) {
00044
00045
         ctl_t ctl;
00046
00047
         static char filename[LEN];
00048
00049
         static double r, t0, z0, z1, dataLon[EX], dataLat[EY], dataZ[EP],
00050
           u0, u1, alpha;
00051
00052
         static float *dataT, *dataU, *dataV, *dataW;
00053
00054
         static int ncid, dims[4], timid, levid, latid, lonid, tid, uid, vid, wid,
00055
            idx, ix, iy, iz, nx, ny, nz, year, mon, day, hour, min, sec;
00056
          /* Allocate... */
00057
00058
         ALLOC(dataT, float,
00059
                  EP * EY * EX);
          ALLOC(dataU, float,
00060
00061
                 EP * EY * EX);
00062
          ALLOC(dataV, float,
00063
                 EP * EY * EX);
          ALLOC(dataW, float,
00064
                 EP * EY * EX);
00065
00066
00067
          /* Check arguments... */
00068
          if (argc < 3)</pre>
00069
            ERRMSG("Give parameters: <ctl> <metbase>");
00070
00071
          /* Read control parameters... */
          read_ctl(argv[1], argc, argv, &ctl);
         read_ctl(argv[1], argc, argv, &ctl);
t0 = scan_ctl(argv[1], argc, argv, "WIND_TO", -1, "0", NULL);
nx = (int) scan_ctl(argv[1], argc, argv, "WIND_NX", -1, "360", NULL);
ny = (int) scan_ctl(argv[1], argc, argv, "WIND_NY", -1, "181", NULL);
nz = (int) scan_ctl(argv[1], argc, argv, "WIND_NZ", -1, "61", NULL);
z0 = scan_ctl(argv[1], argc, argv, "WIND_ZO", -1, "0", NULL);
z1 = scan_ctl(argv[1], argc, argv, "WIND_Z1", -1, "60", NULL);
u0 = scan_ctl(argv[1], argc, argv, "WIND_UO", -1, "38.587660177302", NULL);
u1 = scan_ctl(argv[1], argc, argv, "WIND_U1", -1, "38.587660177302", NULL);
alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0 0", NULL);
00073
00074
00075
00076
00077
00078
00079
00080
          alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);
00081
00082
00083
          /* Check dimensions... */
         if (nx < 1 || nx > EX)
00084
00085
            ERRMSG("Set 1 <= NX <= MAX!");</pre>
            f (ny < 1 || ny > EY)
ERRMSG("Set 1 <= NY <= MAX!");
00086
00087
         if (nz < 1 || nz > EP)
   ERRMSG("Set 1 <= NZ <= MAX!");</pre>
00088
00089
00090
00091
          /* Get time... */
00092
           jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
00093
          t0 = year * 10000. + mon * 100. + day + hour / 24.;
00094
          00095
00096
00097
           /* Create netCDF file...
00098
00099
          NC(nc_create(filename, NC_CLOBBER, &ncid));
00100
         /* Create dimensions... */
NC(nc_def_dim(ncid, "time", 1, &dims[0]));
00101
00102
```

```
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]));
00104
00105
00106
           /* 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));
00107
00108
00110
           NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[3], &lonid));
00111
           NC(nc_def_var(ncid, "T", NC_FLOAT, 4, &dims[0], &tid));
NC(nc_def_var(ncid, "U", NC_FLOAT, 4, &dims[0], &uid));
NC(nc_def_var(ncid, "V", NC_FLOAT, 4, &dims[0], &vid));
00112
00113
00114
           NC(nc_def_var(ncid, "W", NC_FLOAT, 4, &dims[0], &wid));
00115
00116
00117
           /* 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");
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
           add_text_attribute(ncid, "U", "long_name", "U velocity");
add_text_attribute(ncid, "U", "units", "m s**-1");
00128
00129
           add_text_attribute(ncid, "V", "long_name", "V velocity");
00130
           add_text_attribute(ncid, "V", "units", "m s**-1");
add_text_attribute(ncid, "W", "long_name", "Vertical velocity");
add_text_attribute(ncid, "W", "units", "Pa s**-1");
00131
00132
00133
00134
00135
           /* End definition... */
00136
          NC(nc_enddef(ncid));
00137
           /* Set coordinates... */
00138
00139
           for (ix = 0; ix < nx; ix++)
             dataLon[ix] = 360.0 / nx * (double) ix;
00141
           for (iy = 0; iy < ny; iy++)
00142
              dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
00143
           for (iz = 0; iz < nz; iz++)
              dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));
00144
00145
00146
           /* Write coordinates... */
           NC(nc_put_var_double(ncid, timid, &t0));
           NC(nc_put_var_double(ncid, levid, dataZ));
00148
00149
           NC(nc_put_var_double(ncid, lonid, dataLon));
00150
           NC(nc_put_var_double(ncid, latid, dataLat));
00151
00152
           /* Create wind fields (Williamson et al., 1992)... */
00153
           for (ix = 0; ix < nx; ix++)
00154
             for (iy = 0; iy < ny; iy++)</pre>
00155
                 for (iz = 0; iz < nz; iz++) {
                    idx = (iz * ny + iy) * nx + ix;
00156
                    dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
00157
                                                     * (cos(dataLat[iy] * M_PI / 180.0)

* cos(alpha * M_PI / 180.0)
00158
00160
                                                         + sin(dataLat[iy] * M_PI / 180.0)
00161
                                                         * cos(dataLon[ix] * M_PI / 180.0)
                                                          * sin(alpha * M_PI / 180.0)));
00162
                    00163
00164
00165
                                                     * sin(alpha * M_PI / 180.0));
00166
00167
00168
           /* Write wind data... */
00169
           NC(nc_put_var_float(ncid, tid, dataT));
00170
           NC(nc_put_var_float(ncid, uid, dataU));
00171
           NC(nc_put_var_float(ncid, vid, dataV));
00172
           NC(nc_put_var_float(ncid, wid, dataW));
00173
00174
           /* Close file... */
00175
          NC(nc_close(ncid));
00176
           /* Free... */
00177
00178
          free(dataT);
00179
           free (dataU);
00180
           free(dataV);
00181
           free (dataW);
00182
00183
           return EXIT SUCCESS;
00184 }
00185
00187
00188 void add_text_attribute(
00189
          int ncid.
```

5.36 wind.c 211

# Index

add_text_attribute	ctl_t, 15
wind.c, 206	csi_z0
atm_basename	ctl_t, 15
ctl_t, 14	csi_z1
atm_bin	ctl_t, 15
ctl_t, 15	ctl_t, 5
atm_dt_out	atm_basename, 14
ctl_t, 14	atm bin, 15
atm_filter	atm_dt_out, 14
ctl_t, 14	atm filter, 14
atm_gpfile	atm_gpfile, 14
ctl_t, 14	balloon, 13
atm_t, 3	csi basename, 15
lat, 4	csi_dt_out, 15
lon, 4	csi_lat0, 16
np, 4	csi lat1, 16
p, 4	csi lon0, 16
q, 4	csi_lon1, 16
time, 4	csi_modmin, 15
up, 4	csi_modifiin, 13
vp, 5	csi_ny, 16
wp, 5	csi_nz, 15
17	csi_nz, 15
balloon	csi_obsmin, 15
ctl_t, 13	
	csi_z0, 15
cart2geo	csi_z1, 15
libtrac.c, 48	direction, 12
libtrac.h, 104	dt_met, 12
center.c, 23	dt_mod, 12
main, 23	ens_basename, 19
cluster.c, 27	grid_basename, 16
main, 27	grid_dt_out, 16
csi_basename	grid_gpfile, 16
ctl_t, 15	grid_lat0, 17
csi_dt_out	grid_lat1, 18
ctl_t, 15	grid_lon0, 17
csi_lat0	grid_lon1, 17
ctl_t, 16	grid_nx, 17
csi_lat1	grid_ny, 17
ctl_t, 16	grid_nz, 17
csi_lon0	grid_sparse, 17
ctl_t, 16	grid_z0, 17
csi_lon1	grid_z1, <mark>17</mark>
ctl_t, 16	gw_basename, 14
csi_modmin	isosurf, 13
ctl_t, 15	met_np, 12
csi_nx	met_p, 13
ctl_t, 16	met_stage, 13
csi_ny	nq, 9
ctl_t, 16	prof_basename, 18
csi_nz	prof_lat0, 19
ctl_t, 15	prof_lat1, 19
csi_obsfile	prof_lon0, 18
ctl_t, 15	prof_lon1, 18
csi_obsmin	prof_nx, 18

prof_ny, 19	dt_met
prof_nz, 18	ctl_t, 12
prof_obsfile, 18	dt_mod
prof_z0, 18	ctl_t, 12
prof_z1, 18	dx2deg
psc_h2o, 14	libtrac.c, 48
psc_hno3, 14	libtrac.h, 105
qnt_ens, 9	dy2deg
qnt_format, 9	libtrac.c, 49
qnt_gw_sso, 12	libtrac.h, 105
qnt gw u750, 11	dz2dp
qnt_gw_v750, 12	libtrac.c, 49
qnt_gw_var, 12	libtrac.h, 105
qnt_h2o, 11	,
qnt_m, 10	ens_basename
qnt_name, 9	ctl_t, 19
qnt o3, 11	extract.c, 38
qnt_50, 11 qnt_p, 10	main, 39
• —	main, oo
qnt_ps, 10	geo2cart
qnt_pv, 11	libtrac.c, 49
qnt_r, 10	libtrac.h, 106
qnt_rho, 10	get_met
qnt_stat, 11	libtrac.c, 49
qnt_t, 10	,
qnt_theta, 11	libtrac.h, 106
qnt_tice, 11	get_met_help
qnt_tnat, 11	libtrac.c, 50
qnt_tsts, 11	libtrac.h, 107
qnt_u, 10	grid_basename
qnt_unit, 9	ctl_t, 16
qnt_v, 10	grid_dt_out
qnt_w, 10	ctl_t, 16
stat_basename, 19	grid_gpfile
stat_lat, 19	ctl_t, 16
stat_lon, 19	grid_lat0
stat r, 19	ctl_t, 17
t start, 12	grid_lat1
t stop, 12	ctl_t, 18
tdec strat, 14	grid_lon0
tdec trop, 14	ctl_t, 17
turb_dx_strat, 13	grid_lon1
turb_dx_trop, 13	ctl_t, 17
turb_dz_strat, 13	grid_nx
turb dz trop, 13	ctl_t, 17
turb meso, 13	grid ny
turb_meso, 10	ctl_t, 17
deg2dx	grid_nz
libtrac.c, 48	ctl_t, 17
libtrac.h, 104	grid_sparse
deg2dy	ctl_t, 17
libtrac.c, 48	grid_z0
libtrac.h, 105	ctl_t, 17
direction	
ctl_t, 12	grid_z1
dist.c, 33	ctl_t, 17
	gw_basename
main, 33	ctl_t, 14
dp2dz	h0-
libtrac.c, 48	h2o
libtrac.h, 105	met_t, 22

init.c, 41	write_grid, 72		
main, 41	write_prof, 74		
init_simtime	write_station, 77		
trac.c, 172	libtrac.h, 102		
intpol_met_2d	cart2geo, 104		
libtrac.c, 51	deg2dx, 104		
libtrac.h, 107	deg2dy, 105		
intpol_met_3d	dp2dz, 105		
libtrac.c, 51	dx2deg, 105		
libtrac.h, 108	dy2deg, 105		
intpol_met_space	dz2dp, 105		
libtrac.c, 52	geo2cart, 106		
libtrac.h, 108	get_met, 106		
intpol_met_time	get_met_help, 107		
libtrac.c, 53	intpol_met_2d, 107		
libtrac.h, 109	intpol_met_3d, 108		
isosurf	intpol_met_space, 108		
ctl_t, 13	intpol_met_time, 109		
	jsec2time, 110		
jsec2time	locate, 110		
libtrac.c, 54	read_atm, 111		
libtrac.h, 110	read_ctl, 112		
jsec2time.c, 45	read_met, 115		
main, 45	read_met_extrapolate, 117		
	read_met_help, 118		
lat	read_met_ml2pl, 118		
atm_t, 4	read_met_periodic, 119		
met_t, 21	scan_ctl, 119		
libtrac.c, 46	time2jsec, 120		
cart2geo, 48	timer, 121		
deg2dx, 48	tropopause, 121		
deg2dy, 48	write_atm, 123		
dp2dz, 48	write_csi, 125		
dx2deg, 48	write_ens, 127		
dy2deg, 49	write_grid, 129		
dz2dp, 49	write_prof, 131		
geo2cart, 49	write_station, 133		
get_met, 49	locate		
get_met_help, 50	libtrac.c, 54		
intpol_met_2d, 51	libtrac.h, 110		
intpol_met_3d, 51	lon		
intpol_met_space, 52	atm_t, 4		
intpol_met_time, 53	met_t, 21		
jsec2time, 54			
locate, 54	main		
read_atm, 55	center.c, 23		
read_ctl, 56	cluster.c, 27		
read_met, 59	dist.c, 33		
read_met_extrapolate, 61	extract.c, 39		
read_met_help, 62	init.c, 41		
read_met_nariadia_63	jsec2time.c, 45		
read_met_periodic, 63	match.c, 142		
scan_ctl, 63	met_map.c, 147		
time2jsec, 64 timer, 64	met_prof.c, 150		
	met_sample.c, 155 met zm.c, 158		
tropopause, 65			
write_atm, 67	smago.c, 162 split.c, 166		
write_csi, 68 write ens, 70	time2jsec.c, 170		
WII.6_6113, 70	ume2j360.0, 170		

trac.c, 186		met_t, 21
wind.c, 206	ny	
match.c, 142		met_t, 21
main, 142		
met_map.c, 146	о3	
main, 147		met_t, 22
met_np		
ctl_t, 12	р	-4 4 4
met_p		atm_t, 4
ctl_t, 13	nl	met_t, 21
met_prof.c, 150	pl	mot t 22
main, 150	nrof	met_t, <mark>22</mark> _basename
met_sample.c, 154	pioi	ctl_t, 18
main, 155	nrof	_lat0
met_stage	proi	_iato ctl_t, 19
ctl_t, 13 met t, 20	prof	_lat1
h2o, 22	<b>J</b>	ctl_t, 19
lat, 21	prof	_lon0
lon, 21	•	 ctl_t, 18
np, 21	prof	_lon1
nx, 21	•	 ctl_t, 18
ny, 21	prof	_nx
03, 22		ctl_t, 18
p, 21	prof	_ny
pl, 22		ctl_t, 19
ps, 21	prof	_nz
t, <mark>22</mark>		ctl_t, 18
time, 21	prof	_obsfile
u, 22		ctl_t, 18
v, 22	prof	
w, 22		ctl_t, 18
met_zm.c, 157	prof	
main, 158		ctl_t, 18
module_advection	ps	
trac.c, 172		met_t, 21
module_decay	psc_	_h2o
trac.c, 172	nco	ctl_t, 14 _hno3
module_diffusion_meso	μου	ctl_t, 14
trac.c, 173 module diffusion turb		σιι <u>_</u> ι, 1 <del>-1</del>
trac.c, 175	q	
module isosurf	-1	atm t, 4
trac.c, 176	qnt_	_ens
module meteo		ctl_t, 9
trac.c, 178	qnt_	_format
module_meteo_hno3		ctl_t, 9
trac.c, 180	qnt_	_gw_sso
module_position		ctl_t, 12
trac.c, 184	qnt_	_gw_u750
module_sedi		ctl_t, 11
trac.c, 184	qnt_	_gwv750
		ctl_t, 12
np	qnt_	_gw_var
atm_t, 4	-	ctl_t, 12
met_t, 21	qnt_	_h2o
nq	A I	ctl_t, 11
ctl_t, 9	qnt_	
nx		ctl_t, 10

qnt_name	libtrac.h, 119
ctl_t, 9	smago.c, 162
qnt_o3	main, 162
ctl_t, 11	split.c, 165
ant b	main, 166
ctl_t, 10	stat basename
qnt_ps	ctl_t, 19
ctl_t, 10	stat lat
	<del>-</del> .
qnt_pv	ctl_t, 19
ctl_t, 11	stat_lon
qnt_r	ctl_t, 19
ctl_t, 10	stat_r
qnt_rho	ctl_t, 19
ctl_t, 10	
qnt_stat	t
ctl_t, 11	met_t, 22
qnt_t	t_start
ctl t, 10	ctl_t, 12
gnt theta	t_stop
ctl t, 11	ctl_t, 12
	tdec strat
qnt_tice	ctl_t, 14
ctl_t, 11	tdec trop
qnt_tnat	<del>-</del> ·
ctl_t, 11	ctl_t, 14
qnt_tsts	time
ctl_t, 11	atm_t, 4
qnt_u	met_t, 21
ctl_t, 10	time2jsec
qnt_unit	libtrac.c, 64
ctl t, 9	libtrac.h, 120
<del>-</del> '	time2jsec.c, 169
qnt_v	main, 170
ctl_t, 10	timer
qnt_w	libtrac.c, 64
ctl_t, 10	libtrac.h, 121
read_atm	trac.c, 171
libtrac.c, 55	init_simtime, 172
libtrac.h, 111	main, 186
read_ctl	module_advection, 172
libtrac.c, 56	module_decay, 172
libtrac.h, 112	module_diffusion_meso, 173
read_met	module_diffusion_turb, 175
libtrac.c, 59	module_isosurf, 176
libtrac.h, 115	module meteo, 178
read_met_extrapolate	module_meteo_hno3, 180
libtrac.c, 61	module_position, 184
•	module_sedi, 184
libtrac.h, 117	write_output, 185
read_met_help	_ ·
libtrac.c, 62	tropopause
libtrac.h, 118	libtrac.c, 65
read_met_ml2pl	libtrac.h, 121
libtrac.c, 62	turb_dx_strat
libtrac.h, 118	ctl_t, 13
read_met_periodic	turb_dx_trop
libtrac.c, 63	ctl_t, 13
libtrac.h, 119	turb_dz_strat
,	 ctl_t, 13
scan_ctl	turb_dz_trop
libtrac.c, 63	ctl_t, 13
	<u>-</u> -,

```
turb_meso
    ctl_t, 13
u
    met_t, 22
up
     atm_t, 4
٧
     met_t, 22
vp
     atm_t, 5
     met_t, 22
wind.c, 206
     add_text_attribute, 206
     main, 206
wp
     atm_t, 5
write_atm
    libtrac.c, 67
    libtrac.h, 123
write_csi
     libtrac.c, 68
    libtrac.h, 125
write_ens
    libtrac.c, 70
    libtrac.h, 127
write_grid
    libtrac.c, 72
    libtrac.h, 129
write_output
     trac.c, 185
write_prof
    libtrac.c, 74
    libtrac.h, 131
write_station
    libtrac.c, 77
    libtrac.h, 133
```