

MPTRAC

Generated by Doxygen 1.8.17

<b>1 Main Page</b>	<b>1</b>
<b>2 Data Structure Index</b>	<b>1</b>
2.1 Data Structures	1
<b>3 File Index</b>	<b>2</b>
3.1 File List	2
<b>4 Data Structure Documentation</b>	<b>3</b>
4.1 atm_t Struct Reference	3
4.1.1 Detailed Description	3
4.1.2 Field Documentation	3
4.2 cache_t Struct Reference	4
4.2.1 Detailed Description	5
4.2.2 Field Documentation	5
4.3 ctl_t Struct Reference	7
4.3.1 Detailed Description	13
4.3.2 Field Documentation	13
4.4 met_t Struct Reference	37
4.4.1 Detailed Description	38
4.4.2 Field Documentation	38
<b>5 File Documentation</b>	<b>44</b>
5.1 atm_conv.c File Reference	44
5.1.1 Detailed Description	44
5.1.2 Function Documentation	44
5.2 atm_conv.c	45
5.3 atm_dist.c File Reference	46
5.3.1 Detailed Description	46
5.3.2 Function Documentation	46
5.4 atm_dist.c	51
5.5 atm_init.c File Reference	55
5.5.1 Detailed Description	55
5.5.2 Function Documentation	55
5.6 atm_init.c	57
5.7 atm_select.c File Reference	59
5.7.1 Detailed Description	59
5.7.2 Function Documentation	59
5.8 atm_select.c	61
5.9 atm_split.c File Reference	63
5.9.1 Detailed Description	63
5.9.2 Function Documentation	63
5.10 atm_split.c	65
5.11 atm_stat.c File Reference	67

5.11.1 Detailed Description . . . . .	68
5.11.2 Function Documentation . . . . .	68
5.12 atm_stat.c . . . . .	70
5.13 day2doy.c File Reference . . . . .	73
5.13.1 Detailed Description . . . . .	73
5.13.2 Function Documentation . . . . .	73
5.14 day2doy.c . . . . .	74
5.15 doy2day.c File Reference . . . . .	75
5.15.1 Detailed Description . . . . .	75
5.15.2 Function Documentation . . . . .	75
5.16 doy2day.c . . . . .	76
5.17 jsec2time.c File Reference . . . . .	76
5.17.1 Detailed Description . . . . .	76
5.17.2 Function Documentation . . . . .	76
5.18 jsec2time.c . . . . .	77
5.19 lapse.c File Reference . . . . .	78
5.19.1 Detailed Description . . . . .	78
5.19.2 Function Documentation . . . . .	78
5.20 lapse.c . . . . .	81
5.21 libtrac.c File Reference . . . . .	84
5.21.1 Detailed Description . . . . .	86
5.21.2 Function Documentation . . . . .	86
5.22 libtrac.c . . . . .	142
5.23 libtrac.h File Reference . . . . .	201
5.23.1 Function Documentation . . . . .	203
5.24 libtrac.h . . . . .	260
5.25 met_map.c File Reference . . . . .	273
5.25.1 Detailed Description . . . . .	273
5.25.2 Function Documentation . . . . .	274
5.26 met_map.c . . . . .	277
5.27 met_prof.c File Reference . . . . .	280
5.27.1 Detailed Description . . . . .	280
5.27.2 Function Documentation . . . . .	280
5.28 met_prof.c . . . . .	283
5.29 met_sample.c File Reference . . . . .	286
5.29.1 Detailed Description . . . . .	286
5.29.2 Function Documentation . . . . .	286
5.30 met_sample.c . . . . .	288
5.31 met_spec.c File Reference . . . . .	290
5.31.1 Detailed Description . . . . .	291
5.31.2 Function Documentation . . . . .	291
5.32 met_spec.c . . . . .	293

5.33 met_zm.c File Reference . . . . .	295
5.33.1 Detailed Description . . . . .	295
5.33.2 Function Documentation . . . . .	295
5.34 met_zm.c . . . . .	298
5.35 time2jsec.c File Reference . . . . .	301
5.35.1 Detailed Description . . . . .	301
5.35.2 Function Documentation . . . . .	301
5.36 time2jsec.c . . . . .	302
5.37 trac.c File Reference . . . . .	303
5.37.1 Detailed Description . . . . .	304
5.37.2 Function Documentation . . . . .	304
5.37.3 Variable Documentation . . . . .	325
5.38 trac.c . . . . .	326
5.39 tropo.c File Reference . . . . .	342
5.39.1 Detailed Description . . . . .	342
5.39.2 Function Documentation . . . . .	342
5.40 tropo.c . . . . .	347
5.41 tropo_sample.c File Reference . . . . .	351
5.41.1 Detailed Description . . . . .	351
5.41.2 Function Documentation . . . . .	351
5.42 tropo_sample.c . . . . .	356
<b>Index</b>	<b>361</b>

## 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:

<https://github.com/slcs-jsc/mptrac>

## 2 Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<b>atm_t</b>	
<b>Atmospheric data</b>	<b>3</b>
<b>cache_t</b>	
<b>Cache data</b>	<b>4</b>

<a href="#">ctl_t</a>	
Control parameters	7
<a href="#">met_t</a>	
Meteorological data	37

## 3 File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

<a href="#">atm_conv.c</a>	44
<a href="#">atm_dist.c</a>	46
<a href="#">atm_init.c</a>	55
<a href="#">atm_select.c</a>	59
<a href="#">atm_split.c</a>	63
<a href="#">atm_stat.c</a>	67
<a href="#">day2doy.c</a>	73
<a href="#">doy2day.c</a>	75
<a href="#">jsec2time.c</a>	76
<a href="#">lapse.c</a>	78
<a href="#">libtrac.c</a>	84
<a href="#">libtrac.h</a>	201
<a href="#">met_map.c</a>	273
<a href="#">met_prof.c</a>	280
<a href="#">met_sample.c</a>	286
<a href="#">met_spec.c</a>	290
<a href="#">met_zm.c</a>	295
<a href="#">time2jsec.c</a>	301
<a href="#">trac.c</a>	303
<a href="#">tropo.c</a>	342
<a href="#">tropo_sample.c</a>	351

## 4 Data Structure Documentation

### 4.1 atm\_t Struct Reference

Atmospheric data.

```
#include <libtrac.h>
```

#### Data Fields

- int `np`  
*Number of air parcels.*
- double `time` [NP]  
*Time [s].*
- double `p` [NP]  
*Pressure [hPa].*
- double `lon` [NP]  
*Longitude [deg].*
- double `lat` [NP]  
*Latitude [deg].*
- double `q` [NQ][NP]  
*Quantity data (for various, user-defined attributes).*

#### 4.1.1 Detailed Description

Atmospheric data.

Definition at line 898 of file `libtrac.h`.

#### 4.1.2 Field Documentation

##### 4.1.2.1 `np` int atm\_t::np

Number of air parcels.

Definition at line 901 of file `libtrac.h`.

##### 4.1.2.2 `time` double atm\_t::time[NP]

Time [s].

Definition at line 904 of file `libtrac.h`.

**4.1.2.3 p** `double atm_t::p[NP]`

Pressure [hPa].

Definition at line [907](#) of file [libtrac.h](#).

**4.1.2.4 lon** `double atm_t::lon[NP]`

Longitude [deg].

Definition at line [910](#) of file [libtrac.h](#).

**4.1.2.5 lat** `double atm_t::lat[NP]`

Latitude [deg].

Definition at line [913](#) of file [libtrac.h](#).

**4.1.2.6 q** `double atm_t::q[NQ] [NP]`

Quantity data (for various, user-defined attributes).

Definition at line [916](#) of file [libtrac.h](#).

The documentation for this struct was generated from the following file:

- [libtrac.h](#)

## 4.2 cache\_t Struct Reference

Cache data.

```
#include <libtrac.h>
```

### Data Fields

- float `up` [NP]  
*Zonal wind perturbation [m/s].*
- float `vp` [NP]  
*Meridional wind perturbation [m/s].*
- float `wv` [NP]  
*Vertical velocity perturbation [hPa/s].*
- double `iso_var` [NP]  
*Isosurface variables.*
- double `iso_ps` [NP]  
*Isosurface balloon pressure [hPa].*
- double `iso_ts` [NP]  
*Isosurface balloon time [s].*
- int `iso_n`  
*Isosurface balloon number of data points.*
- double `tsig` [EX][EY][EP]  
*Cache for reference time of wind standard deviations.*
- float `usig` [EX][EY][EP]  
*Cache for zonal wind standard deviations.*
- float `vsig` [EX][EY][EP]  
*Cache for meridional wind standard deviations.*
- float `wsig` [EX][EY][EP]  
*Cache for vertical velocity standard deviations.*

#### 4.2.1 Detailed Description

Cache data.

Definition at line 921 of file `libtrac.h`.

#### 4.2.2 Field Documentation

##### 4.2.2.1 `up` float `cache_t::up` [NP]

Zonal wind perturbation [m/s].

Definition at line 924 of file `libtrac.h`.

##### 4.2.2.2 `vp` float `cache_t::vp` [NP]

Meridional wind perturbation [m/s].

Definition at line 927 of file `libtrac.h`.



**4.2.2.3 wp** `float cache_t::wp[NP]`

Vertical velocity perturbation [hPa/s].

Definition at line [930](#) of file [libtrac.h](#).

**4.2.2.4 iso\_var** `double cache_t::iso_var[NP]`

Isosurface variables.

Definition at line [933](#) of file [libtrac.h](#).

**4.2.2.5 iso\_ps** `double cache_t::iso_ps[NP]`

Isosurface balloon pressure [hPa].

Definition at line [936](#) of file [libtrac.h](#).

**4.2.2.6 iso\_ts** `double cache_t::iso_ts[NP]`

Isosurface balloon time [s].

Definition at line [939](#) of file [libtrac.h](#).

**4.2.2.7 iso\_n** `int cache_t::iso_n`

Isosurface balloon number of data points.

Definition at line [942](#) of file [libtrac.h](#).

**4.2.2.8 tsig** `double cache_t::tsig[EX][EY][EP]`

Cache for reference time of wind standard deviations.

Definition at line [945](#) of file [libtrac.h](#).

**4.2.2.9** `usig` `float cache_t::usig[EX][EY][EP]`

Cache for zonal wind standard deviations.

Definition at line 948 of file [libtrac.h](#).

**4.2.2.10** `vsig` `float cache_t::vsig[EX][EY][EP]`

Cache for meridional wind standard deviations.

Definition at line 951 of file [libtrac.h](#).

**4.2.2.11** `wsig` `float cache_t::wsig[EX][EY][EP]`

Cache for vertical velocity standard deviations.

Definition at line 954 of file [libtrac.h](#).

The documentation for this struct was generated from the following file:

- [libtrac.h](#)

**4.3** `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\\_ts](#)  
*Quantity array index for surface temperature.*
- int [qnt\\_zs](#)  
*Quantity array index for surface geopotential height.*
- int [qnt\\_us](#)  
*Quantity array index for surface zonal wind.*
- int [qnt\\_vs](#)  
*Quantity array index for surface meridional wind.*
- int [qnt\\_pt](#)  
*Quantity array index for tropopause pressure.*
- int [qnt\\_tt](#)  
*Quantity array index for tropopause temperature.*
- int [qnt\\_zt](#)  
*Quantity array index for tropopause geopotential height.*
- int [qnt\\_h2ot](#)  
*Quantity array index for tropopause water vapor vmr.*
- int [qnt\\_z](#)  
*Quantity array index for geopotential height.*
- int [qnt\\_p](#)  
*Quantity array index for pressure.*
- int [qnt\\_t](#)  
*Quantity array index for temperature.*
- int [qnt\\_u](#)  
*Quantity array index for zonal wind.*
- int [qnt\\_v](#)  
*Quantity array index for meridional wind.*
- int [qnt\\_w](#)  
*Quantity array index for vertical velocity.*
- int [qnt\\_h2o](#)  
*Quantity array index for water vapor vmr.*
- int [qnt\\_o3](#)  
*Quantity array index for ozone vmr.*
- int [qnt\\_lwc](#)  
*Quantity array index for cloud liquid water content.*
- int [qnt\\_iwc](#)  
*Quantity array index for cloud ice water content.*
- int [qnt\\_pc](#)  
*Quantity array index for cloud top pressure.*
- int [qnt\\_cl](#)  
*Quantity array index for total column cloud water.*
- int [qnt\\_plcl](#)  
*Quantity array index for pressure at lifted condensation level (LCL).*
- int [qnt\\_plfc](#)  
*Quantity array index for pressure at level of free convection (LCF).*
- int [qnt\\_pel](#)  
*Quantity array index for pressure at equilibrium level (EL).*
- int [qnt\\_cape](#)  
*Quantity array index for convective available potential energy (CAPE).*
- int [qnt\\_hno3](#)

- Quantity array index for nitric acid vmr.*

  - int `qnt_oh`
- Quantity array index for hydroxyl number concentrations.*

  - int `qnt_psat`
- Quantity array index for saturation pressure over water.*

  - int `qnt_psice`
- Quantity array index for saturation pressure over ice.*

  - int `qnt_pw`
- Quantity array index for partial water vapor pressure.*

  - int `qnt_sh`
- Quantity array index for specific humidity.*

  - int `qnt_rh`
- Quantity array index for relative humidity over water.*

  - int `qnt_rhice`
- Quantity array index for relative humidity over ice.*

  - int `qnt_theta`
- Quantity array index for potential temperature.*

  - int `qnt_tvirt`
- Quantity array index for virtual temperature.*

  - int `qnt_lapse`
- Quantity array index for lapse rate.*

  - int `qnt_vh`
- Quantity array index for horizontal wind.*

  - int `qnt_vz`
- Quantity array index for vertical velocity.*

  - int `qnt_pv`
- Quantity array index for potential vorticity.*

  - int `qnt_tdew`
- Quantity array index for dew point temperature.*

  - 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 `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].*

  - char `metbase` [LEN]
- Basename for meteorological data.*

  - double `dt_met`
- Time step of meteorological data [s].*

  - int `met_dx`
- Stride for longitudes.*

- int `met_dy`  
*Stride for latitudes.*
- int `met_dp`  
*Stride for pressure levels.*
- int `met_sx`  
*Smoothing for longitudes.*
- int `met_sy`  
*Smoothing for latitudes.*
- int `met_sp`  
*Smoothing for pressure levels.*
- double `met_detrend`  
*FWHM of horizontal Gaussian used for detrending [km].*
- int `met_np`  
*Number of target pressure levels.*
- double `met_p` [EP]  
*Target pressure levels [hPa].*
- int `met_geopot_sx`  
*Longitudinal smoothing of geopotential heights.*
- int `met_geopot_sy`  
*Latitudinal smoothing of geopotential heights.*
- int `met_tropo`  
*Tropopause definition (0=none, 1=clim, 2=cold point, 3=WMO\_1st, 4=WMO\_2nd).*
- double `met_dt_out`  
*Time step for sampling of meteo data along trajectories [s].*
- int `isosurf`  
*Isosurface parameter (0=none, 1=pressure, 2=density, 3=theta, 4=balloon).*
- char `balloon` [LEN]  
*Balloon position filename.*
- double `turb_dx_trop`  
*Horizontal turbulent diffusion coefficient (troposphere) [ $\text{m}^2/\text{s}$ ].*
- double `turb_dx_strat`  
*Horizontal turbulent diffusion coefficient (stratosphere) [ $\text{m}^2/\text{s}$ ].*
- double `turb_dz_trop`  
*Vertical turbulent diffusion coefficient (troposphere) [ $\text{m}^2/\text{s}$ ].*
- double `turb_dz_strat`  
*Vertical turbulent diffusion coefficient (stratosphere) [ $\text{m}^2/\text{s}$ ].*
- double `turb_mesox`  
*Horizontal scaling factor for mesoscale wind fluctuations.*
- double `turb_mesoz`  
*Vertical scaling factor for mesoscale wind fluctuations.*
- double `conv_cape`  
*CAPE threshold for convection module [J/kg].*
- char `species` [LEN]  
*Species.*
- double `molmass`  
*Molar mass [g/mol].*
- double `tdec_trop`  
*Life time of particles (troposphere) [s].*
- double `tdec_strat`  
*Life time of particles (stratosphere) [s].*
- double `oh_chem` [4]

- Coefficients for OH chemistry ( $k_0$ ,  $n$ ,  $kinf$ ,  $m$ ).*

  - double [dry\\_depo](#) [1]

*Coefficients for dry deposition ( $v$ ).*

  - double [wet\\_depo](#) [8]

*Coefficients for wet deposition ( $A_i$ ,  $B_i$ ,  $H_i$ ,  $C_i$ ,  $A_b$ ,  $B_b$ ,  $H_b$ ,  $C_b$ ).*

  - double [psc\\_h2o](#)

*H<sub>2</sub>O volume mixing ratio for PSC analysis.*

  - double [psc\\_hno3](#)

*HNO<sub>3</sub> volume mixing ratio for PSC analysis.*

  - char [atm\\_basename](#) [LEN]

*Baseline 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\\_stride](#)

*Particle index stride for atmospheric data files.*

  - int [atm\\_type](#)

*Type of atmospheric data files (0=ASCII, 1=binary, 2=netCDF).*

  - char [csi\\_basename](#) [LEN]

*Baseline 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 [ $\text{kg/m}^2$ ].*

  - int [csi\\_nz](#)

*Number of altitudes of gridded CSI data.*

  - double [csi\\_z0](#)

*Lower altitude of gridded CSI data [km].*

  - double [csi\\_z1](#)

*Upper altitude of gridded CSI data [km].*

  - int [csi\\_nx](#)

*Number of longitudes of gridded CSI data.*

  - double [csi\\_lon0](#)

*Lower longitude of gridded CSI data [deg].*

  - double [csi\\_lon1](#)

*Upper longitude of gridded CSI data [deg].*

  - int [csi\\_ny](#)

*Number of latitudes of gridded CSI data.*

  - double [csi\\_lat0](#)

*Lower latitude of gridded CSI data [deg].*

  - double [csi\\_lat1](#)

*Upper latitude of gridded CSI data [deg].*

  - char [grid\\_basename](#) [LEN]

*Baseline 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 `sample_basename` [LEN]  
*Basename of sample data file.*
- char `sample_obsfile` [LEN]

- Observation data file for sample output.*
  - double `sample_dx`  
*Horizontal radius for sample output [km].*
  - double `sample_dz`  
*Layer width for sample output [km].*
  - char `stat_basename` [LEN]  
*Basename of station data file.*
  - double `stat_lon`  
*Longitude of station [deg].*
  - double `stat_lat`  
*Latitude of station [deg].*
  - double `stat_r`  
*Search radius around station [km].*

#### 4.3.1 Detailed Description

Control parameters.

Definition at line 465 of file `libtrac.h`.

#### 4.3.2 Field Documentation

##### 4.3.2.1 `nq` `int` `ctl_t::nq`

Number of quantities.

Definition at line 468 of file `libtrac.h`.

##### 4.3.2.2 `qnt_name` `char` `ctl_t::qnt_name[NQ]` [LEN]

Quantity names.

Definition at line 471 of file `libtrac.h`.

##### 4.3.2.3 `qnt_unit` `char` `ctl_t::qnt_unit[NQ]` [LEN]

Quantity units.

Definition at line 474 of file `libtrac.h`.



**4.3.2.4 qnt\_format** `char ctl_t::qnt_format[NQ][LEN]`

Quantity output format.

Definition at line [477](#) of file [libtrac.h](#).

**4.3.2.5 qnt\_ens** `int ctl_t::qnt_ens`

Quantity array index for ensemble IDs.

Definition at line [480](#) of file [libtrac.h](#).

**4.3.2.6 qnt\_m** `int ctl_t::qnt_m`

Quantity array index for mass.

Definition at line [483](#) of file [libtrac.h](#).

**4.3.2.7 qnt\_rho** `int ctl_t::qnt_rho`

Quantity array index for particle density.

Definition at line [486](#) of file [libtrac.h](#).

**4.3.2.8 qnt\_r** `int ctl_t::qnt_r`

Quantity array index for particle radius.

Definition at line [489](#) of file [libtrac.h](#).

**4.3.2.9 qnt\_ps** `int ctl_t::qnt_ps`

Quantity array index for surface pressure.

Definition at line [492](#) of file [libtrac.h](#).

**4.3.2.10** `qnt_ts` `int ctl_t::qnt_ts`

Quantity array index for surface temperature.

Definition at line [495](#) of file [libtrac.h](#).

**4.3.2.11** `qnt_zs` `int ctl_t::qnt_zs`

Quantity array index for surface geopotential height.

Definition at line [498](#) of file [libtrac.h](#).

**4.3.2.12** `qnt_us` `int ctl_t::qnt_us`

Quantity array index for surface zonal wind.

Definition at line [501](#) of file [libtrac.h](#).

**4.3.2.13** `qnt_vs` `int ctl_t::qnt_vs`

Quantity array index for surface meridional wind.

Definition at line [504](#) of file [libtrac.h](#).

**4.3.2.14** `qnt_pt` `int ctl_t::qnt_pt`

Quantity array index for tropopause pressure.

Definition at line [507](#) of file [libtrac.h](#).

**4.3.2.15** `qnt_tt` `int ctl_t::qnt_tt`

Quantity array index for tropopause temperature.

Definition at line [510](#) of file [libtrac.h](#).

**4.3.2.16** `qnt_zt` `int ctl_t::qnt_zt`

Quantity array index for tropopause geopotential height.

Definition at line [513](#) of file [libtrac.h](#).

**4.3.2.17** `qnt_h2ot` `int ctl_t::qnt_h2ot`

Quantity array index for tropopause water vapor vmr.

Definition at line [516](#) of file [libtrac.h](#).

**4.3.2.18** `qnt_z` `int ctl_t::qnt_z`

Quantity array index for geopotential height.

Definition at line [519](#) of file [libtrac.h](#).

**4.3.2.19** `qnt_p` `int ctl_t::qnt_p`

Quantity array index for pressure.

Definition at line [522](#) of file [libtrac.h](#).

**4.3.2.20** `qnt_t` `int ctl_t::qnt_t`

Quantity array index for temperature.

Definition at line [525](#) of file [libtrac.h](#).

**4.3.2.21** `qnt_u` `int ctl_t::qnt_u`

Quantity array index for zonal wind.

Definition at line [528](#) of file [libtrac.h](#).

**4.3.2.22** `qnt_v` `int` `ctl_t::qnt_v`

Quantity array index for meridional wind.

Definition at line [531](#) of file [libtrac.h](#).

**4.3.2.23** `qnt_w` `int` `ctl_t::qnt_w`

Quantity array index for vertical velocity.

Definition at line [534](#) of file [libtrac.h](#).

**4.3.2.24** `qnt_h2o` `int` `ctl_t::qnt_h2o`

Quantity array index for water vapor vmr.

Definition at line [537](#) of file [libtrac.h](#).

**4.3.2.25** `qnt_o3` `int` `ctl_t::qnt_o3`

Quantity array index for ozone vmr.

Definition at line [540](#) of file [libtrac.h](#).

**4.3.2.26** `qnt_lwc` `int` `ctl_t::qnt_lwc`

Quantity array index for cloud liquid water content.

Definition at line [543](#) of file [libtrac.h](#).

**4.3.2.27** `qnt_iwc` `int` `ctl_t::qnt_iwc`

Quantity array index for cloud ice water content.

Definition at line [546](#) of file [libtrac.h](#).

**4.3.2.28 qnt\_pc** `int ctl_t::qnt_pc`

Quantity array index for cloud top pressure.

Definition at line [549](#) of file [libtrac.h](#).

**4.3.2.29 qnt\_cl** `int ctl_t::qnt_cl`

Quantity array index for total column cloud water.

Definition at line [552](#) of file [libtrac.h](#).

**4.3.2.30 qnt\_plcl** `int ctl_t::qnt_plcl`

Quantity array index for pressure at lifted condensation level (LCL).

Definition at line [555](#) of file [libtrac.h](#).

**4.3.2.31 qnt\_plfc** `int ctl_t::qnt_plfc`

Quantity array index for pressure at level of free convection (LCF).

Definition at line [558](#) of file [libtrac.h](#).

**4.3.2.32 qnt\_pel** `int ctl_t::qnt_pel`

Quantity array index for pressure at equilibrium level (EL).

Definition at line [561](#) of file [libtrac.h](#).

**4.3.2.33 qnt\_cape** `int ctl_t::qnt_cape`

Quantity array index for convective available potential energy (CAPE).

Definition at line [564](#) of file [libtrac.h](#).

**4.3.2.34** `qnt_hno3` `int ctl_t::qnt_hno3`

Quantity array index for nitric acid vmr.

Definition at line 567 of file [libtrac.h](#).

**4.3.2.35** `qnt_oh` `int ctl_t::qnt_oh`

Quantity array index for hydroxyl number concentrations.

Definition at line 570 of file [libtrac.h](#).

**4.3.2.36** `qnt_psat` `int ctl_t::qnt_psat`

Quantity array index for saturation pressure over water.

Definition at line 573 of file [libtrac.h](#).

**4.3.2.37** `qnt_psice` `int ctl_t::qnt_psice`

Quantity array index for saturation pressure over ice.

Definition at line 576 of file [libtrac.h](#).

**4.3.2.38** `qnt_pw` `int ctl_t::qnt_pw`

Quantity array index for partial water vapor pressure.

Definition at line 579 of file [libtrac.h](#).

**4.3.2.39** `qnt_sh` `int ctl_t::qnt_sh`

Quantity array index for specific humidity.

Definition at line 582 of file [libtrac.h](#).

**4.3.2.40 qnt\_rh** `int ctl_t::qnt_rh`

Quantity array index for relative humidity over water.

Definition at line [585](#) of file [libtrac.h](#).

**4.3.2.41 qnt\_rhice** `int ctl_t::qnt_rhice`

Quantity array index for relative humidity over ice.

Definition at line [588](#) of file [libtrac.h](#).

**4.3.2.42 qnt\_theta** `int ctl_t::qnt_theta`

Quantity array index for potential temperature.

Definition at line [591](#) of file [libtrac.h](#).

**4.3.2.43 qnt\_tvirt** `int ctl_t::qnt_tvirt`

Quantity array index for virtual temperature.

Definition at line [594](#) of file [libtrac.h](#).

**4.3.2.44 qnt\_lapse** `int ctl_t::qnt_lapse`

Quantity array index for lapse rate.

Definition at line [597](#) of file [libtrac.h](#).

**4.3.2.45 qnt\_vh** `int ctl_t::qnt_vh`

Quantity array index for horizontal wind.

Definition at line [600](#) of file [libtrac.h](#).

**4.3.2.46** `qnt_vz` `int ctl_t::qnt_vz`

Quantity array index for vertical velocity.

Definition at line 603 of file [libtrac.h](#).

**4.3.2.47** `qnt_pv` `int ctl_t::qnt_pv`

Quantity array index for potential vorticity.

Definition at line 606 of file [libtrac.h](#).

**4.3.2.48** `qnt_tdew` `int ctl_t::qnt_tdew`

Quantity array index for dew point temperature.

Definition at line 609 of file [libtrac.h](#).

**4.3.2.49** `qnt_tice` `int ctl_t::qnt_tice`

Quantity array index for T\_ice.

Definition at line 612 of file [libtrac.h](#).

**4.3.2.50** `qnt_tsts` `int ctl_t::qnt_tsts`

Quantity array index for T\_STS.

Definition at line 615 of file [libtrac.h](#).

**4.3.2.51** `qnt_tnat` `int ctl_t::qnt_tnat`

Quantity array index for T\_NAT.

Definition at line 618 of file [libtrac.h](#).



**4.3.2.52 qnt\_stat** `int ctl_t::qnt_stat`

Quantity array index for station flag.

Definition at line [621](#) of file [libtrac.h](#).

**4.3.2.53 direction** `int ctl_t::direction`

Direction flag (1=forward calculation, -1=backward calculation).

Definition at line [624](#) of file [libtrac.h](#).

**4.3.2.54 t\_start** `double ctl_t::t_start`

Start time of simulation [s].

Definition at line [627](#) of file [libtrac.h](#).

**4.3.2.55 t\_stop** `double ctl_t::t_stop`

Stop time of simulation [s].

Definition at line [630](#) of file [libtrac.h](#).

**4.3.2.56 dt\_mod** `double ctl_t::dt_mod`

Time step of simulation [s].

Definition at line [633](#) of file [libtrac.h](#).

**4.3.2.57 metbase** `char ctl_t::metbase[LEN]`

Basename for meteorological data.

Definition at line [636](#) of file [libtrac.h](#).

**4.3.2.58 `dt_met`** `double ctl_t::dt_met`

Time step of meteorological data [s].

Definition at line 639 of file [libtrac.h](#).

**4.3.2.59 `met_dx`** `int ctl_t::met_dx`

Stride for longitudes.

Definition at line 642 of file [libtrac.h](#).

**4.3.2.60 `met_dy`** `int ctl_t::met_dy`

Stride for latitudes.

Definition at line 645 of file [libtrac.h](#).

**4.3.2.61 `met_dp`** `int ctl_t::met_dp`

Stride for pressure levels.

Definition at line 648 of file [libtrac.h](#).

**4.3.2.62 `met_sx`** `int ctl_t::met_sx`

Smoothing for longitudes.

Definition at line 651 of file [libtrac.h](#).

**4.3.2.63 `met_sy`** `int ctl_t::met_sy`

Smoothing for latitudes.

Definition at line 654 of file [libtrac.h](#).

**4.3.2.64 met\_sp** `int ctl_t::met_sp`

Smoothing for pressure levels.

Definition at line [657](#) of file [libtrac.h](#).

**4.3.2.65 met\_detrend** `double ctl_t::met_detrend`

FWHM of horizontal Gaussian used for detrending [km].

Definition at line [660](#) of file [libtrac.h](#).

**4.3.2.66 met\_np** `int ctl_t::met_np`

Number of target pressure levels.

Definition at line [663](#) of file [libtrac.h](#).

**4.3.2.67 met\_p** `double ctl_t::met_p[EP]`

Target pressure levels [hPa].

Definition at line [666](#) of file [libtrac.h](#).

**4.3.2.68 met\_geopot\_sx** `int ctl_t::met_geopot_sx`

Longitudinal smoothing of geopotential heights.

Definition at line [669](#) of file [libtrac.h](#).

**4.3.2.69 met\_geopot\_sy** `int ctl_t::met_geopot_sy`

Latitudinal smoothing of geopotential heights.

Definition at line [672](#) of file [libtrac.h](#).

**4.3.2.70 met\_tropo** `int ctl_t::met_tropo`

Tropopause definition (0=none, 1=clim, 2=cold point, 3=WMO\_1st, 4=WMO\_2nd).

Definition at line 676 of file [libtrac.h](#).

**4.3.2.71 met\_dt\_out** `double ctl_t::met_dt_out`

Time step for sampling of meteo data along trajectories [s].

Definition at line 679 of file [libtrac.h](#).

**4.3.2.72 isosurf** `int ctl_t::isosurf`

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

Definition at line 683 of file [libtrac.h](#).

**4.3.2.73 balloon** `char ctl_t::balloon[LEN]`

Balloon position filename.

Definition at line 686 of file [libtrac.h](#).

**4.3.2.74 turb\_dx\_trop** `double ctl_t::turb_dx_trop`

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

Definition at line 689 of file [libtrac.h](#).

**4.3.2.75 turb\_dx\_strat** `double ctl_t::turb_dx_strat`

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

Definition at line 692 of file [libtrac.h](#).

**4.3.2.76 turb\_dz\_trop** `double ctl_t::turb_dz_trop`

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

Definition at line 695 of file [libtrac.h](#).

**4.3.2.77 turb\_dz\_strat** `double ctl_t::turb_dz_strat`

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

Definition at line 698 of file [libtrac.h](#).

**4.3.2.78 turb\_mesox** `double ctl_t::turb_mesox`

Horizontal scaling factor for mesoscale wind fluctuations.

Definition at line 701 of file [libtrac.h](#).

**4.3.2.79 turb\_mesoz** `double ctl_t::turb_mesoz`

Vertical scaling factor for mesoscale wind fluctuations.

Definition at line 704 of file [libtrac.h](#).

**4.3.2.80 conv\_cape** `double ctl_t::conv_cape`

CAPE threshold for convection module [ $\text{J/kg}$ ].

Definition at line 707 of file [libtrac.h](#).

**4.3.2.81 species** `char ctl_t::species[LEN]`

Species.

Definition at line 710 of file [libtrac.h](#).

**4.3.2.82 `molmass`** `double ctl_t::molmass`

Molar mass [g/mol].

Definition at line 713 of file [libtrac.h](#).

**4.3.2.83 `tdec_trop`** `double ctl_t::tdec_trop`

Life time of particles (troposphere) [s].

Definition at line 716 of file [libtrac.h](#).

**4.3.2.84 `tdec_strat`** `double ctl_t::tdec_strat`

Life time of particles (stratosphere) [s].

Definition at line 719 of file [libtrac.h](#).

**4.3.2.85 `oh_chem`** `double ctl_t::oh_chem[4]`

Coefficients for OH chemistry (k0, n, kinf, m).

Definition at line 722 of file [libtrac.h](#).

**4.3.2.86 `dry_depo`** `double ctl_t::dry_depo[1]`

Coefficients for dry deposition (v).

Definition at line 725 of file [libtrac.h](#).

**4.3.2.87 `wet_depo`** `double ctl_t::wet_depo[8]`

Coefficients for wet deposition (Ai, Bi, Hi, Ci, Ab, Bb, Hb, Cb).

Definition at line 728 of file [libtrac.h](#).

**4.3.2.88 psc\_h2o** `double ctl_t::psc_h2o`

H2O volume mixing ratio for PSC analysis.

Definition at line [731](#) of file [libtrac.h](#).

**4.3.2.89 psc\_hno3** `double ctl_t::psc_hno3`

HNO3 volume mixing ratio for PSC analysis.

Definition at line [734](#) of file [libtrac.h](#).

**4.3.2.90 atm\_basename** `char ctl_t::atm_basename[LEN]`

Basename of atmospheric data files.

Definition at line [737](#) of file [libtrac.h](#).

**4.3.2.91 atm\_gpfile** `char ctl_t::atm_gpfile[LEN]`

Gnuplot file for atmospheric data.

Definition at line [740](#) of file [libtrac.h](#).

**4.3.2.92 atm\_dt\_out** `double ctl_t::atm_dt_out`

Time step for atmospheric data output [s].

Definition at line [743](#) of file [libtrac.h](#).

**4.3.2.93 atm\_filter** `int ctl_t::atm_filter`

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

Definition at line [746](#) of file [libtrac.h](#).

**4.3.2.94** `atm_stride` `int ctl_t::atm_stride`

Particle index stride for atmospheric data files.

Definition at line 749 of file [libtrac.h](#).

**4.3.2.95** `atm_type` `int ctl_t::atm_type`

Type of atmospheric data files (0=ASCII, 1=binary, 2=netCDF).

Definition at line 752 of file [libtrac.h](#).

**4.3.2.96** `csi_basename` `char ctl_t::csi_basename[LEN]`

Basename of CSI data files.

Definition at line 755 of file [libtrac.h](#).

**4.3.2.97** `csi_dt_out` `double ctl_t::csi_dt_out`

Time step for CSI data output [s].

Definition at line 758 of file [libtrac.h](#).

**4.3.2.98** `csi_obsfile` `char ctl_t::csi_obsfile[LEN]`

Observation data file for CSI analysis.

Definition at line 761 of file [libtrac.h](#).

**4.3.2.99** `csi_obsmin` `double ctl_t::csi_obsmin`

Minimum observation index to trigger detection.

Definition at line 764 of file [libtrac.h](#).



**4.3.2.100** **csi\_modmin** `double ctl_t::csi_modmin`

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

Definition at line 767 of file [libtrac.h](#).

**4.3.2.101** **csi\_nz** `int ctl_t::csi_nz`

Number of altitudes of gridded CSI data.

Definition at line 770 of file [libtrac.h](#).

**4.3.2.102** **csi\_z0** `double ctl_t::csi_z0`

Lower altitude of gridded CSI data [km].

Definition at line 773 of file [libtrac.h](#).

**4.3.2.103** **csi\_z1** `double ctl_t::csi_z1`

Upper altitude of gridded CSI data [km].

Definition at line 776 of file [libtrac.h](#).

**4.3.2.104** **csi\_nx** `int ctl_t::csi_nx`

Number of longitudes of gridded CSI data.

Definition at line 779 of file [libtrac.h](#).

**4.3.2.105** **csi\_lon0** `double ctl_t::csi_lon0`

Lower longitude of gridded CSI data [deg].

Definition at line 782 of file [libtrac.h](#).

**4.3.2.106** `csi_lon1` `double` `ctl_t::csi_lon1`

Upper longitude of gridded CSI data [deg].

Definition at line 785 of file [libtrac.h](#).

**4.3.2.107** `csi_ny` `int` `ctl_t::csi_ny`

Number of latitudes of gridded CSI data.

Definition at line 788 of file [libtrac.h](#).

**4.3.2.108** `csi_lat0` `double` `ctl_t::csi_lat0`

Lower latitude of gridded CSI data [deg].

Definition at line 791 of file [libtrac.h](#).

**4.3.2.109** `csi_lat1` `double` `ctl_t::csi_lat1`

Upper latitude of gridded CSI data [deg].

Definition at line 794 of file [libtrac.h](#).

**4.3.2.110** `grid_basename` `char` `ctl_t::grid_basename[LEN]`

Basename of grid data files.

Definition at line 797 of file [libtrac.h](#).

**4.3.2.111** `grid_gpfile` `char` `ctl_t::grid_gpfile[LEN]`

Gnuplot file for gridded data.

Definition at line 800 of file [libtrac.h](#).

**4.3.2.112 grid\_dt\_out** `double ctl_t::grid_dt_out`

Time step for gridded data output [s].

Definition at line [803](#) of file [libtrac.h](#).

**4.3.2.113 grid\_sparse** `int ctl_t::grid_sparse`

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

Definition at line [806](#) of file [libtrac.h](#).

**4.3.2.114 grid\_nz** `int ctl_t::grid_nz`

Number of altitudes of gridded data.

Definition at line [809](#) of file [libtrac.h](#).

**4.3.2.115 grid\_z0** `double ctl_t::grid_z0`

Lower altitude of gridded data [km].

Definition at line [812](#) of file [libtrac.h](#).

**4.3.2.116 grid\_z1** `double ctl_t::grid_z1`

Upper altitude of gridded data [km].

Definition at line [815](#) of file [libtrac.h](#).

**4.3.2.117 grid\_nx** `int ctl_t::grid_nx`

Number of longitudes of gridded data.

Definition at line [818](#) of file [libtrac.h](#).

**4.3.2.118** `grid_lon0` `double` `ctl_t::grid_lon0`

Lower longitude of gridded data [deg].

Definition at line 821 of file [libtrac.h](#).

**4.3.2.119** `grid_lon1` `double` `ctl_t::grid_lon1`

Upper longitude of gridded data [deg].

Definition at line 824 of file [libtrac.h](#).

**4.3.2.120** `grid_ny` `int` `ctl_t::grid_ny`

Number of latitudes of gridded data.

Definition at line 827 of file [libtrac.h](#).

**4.3.2.121** `grid_lat0` `double` `ctl_t::grid_lat0`

Lower latitude of gridded data [deg].

Definition at line 830 of file [libtrac.h](#).

**4.3.2.122** `grid_lat1` `double` `ctl_t::grid_lat1`

Upper latitude of gridded data [deg].

Definition at line 833 of file [libtrac.h](#).

**4.3.2.123** `prof_basename` `char` `ctl_t::prof_basename[LEN]`

Basename for profile output file.

Definition at line 836 of file [libtrac.h](#).

**4.3.2.124 prof\_obsfile** `char ctl_t::prof_obsfile[LEN]`

Observation data file for profile output.

Definition at line 839 of file [libtrac.h](#).

**4.3.2.125 prof\_nz** `int ctl_t::prof_nz`

Number of altitudes of gridded profile data.

Definition at line 842 of file [libtrac.h](#).

**4.3.2.126 prof\_z0** `double ctl_t::prof_z0`

Lower altitude of gridded profile data [km].

Definition at line 845 of file [libtrac.h](#).

**4.3.2.127 prof\_z1** `double ctl_t::prof_z1`

Upper altitude of gridded profile data [km].

Definition at line 848 of file [libtrac.h](#).

**4.3.2.128 prof\_nx** `int ctl_t::prof_nx`

Number of longitudes of gridded profile data.

Definition at line 851 of file [libtrac.h](#).

**4.3.2.129 prof\_lon0** `double ctl_t::prof_lon0`

Lower longitude of gridded profile data [deg].

Definition at line 854 of file [libtrac.h](#).

**4.3.2.130** `prof_lon1` `double` `ctl_t::prof_lon1`

Upper longitude of gridded profile data [deg].

Definition at line 857 of file [libtrac.h](#).

**4.3.2.131** `prof_ny` `int` `ctl_t::prof_ny`

Number of latitudes of gridded profile data.

Definition at line 860 of file [libtrac.h](#).

**4.3.2.132** `prof_lat0` `double` `ctl_t::prof_lat0`

Lower latitude of gridded profile data [deg].

Definition at line 863 of file [libtrac.h](#).

**4.3.2.133** `prof_lat1` `double` `ctl_t::prof_lat1`

Upper latitude of gridded profile data [deg].

Definition at line 866 of file [libtrac.h](#).

**4.3.2.134** `ens_basename` `char` `ctl_t::ens_basename[LEN]`

Basename of ensemble data file.

Definition at line 869 of file [libtrac.h](#).

**4.3.2.135** `sample_basename` `char` `ctl_t::sample_basename[LEN]`

Basename of sample data file.

Definition at line 872 of file [libtrac.h](#).

**4.3.2.136 sample\_obsfile** `char ctl_t::sample_obsfile[LEN]`

Observation data file for sample output.

Definition at line [875](#) of file [libtrac.h](#).

**4.3.2.137 sample\_dx** `double ctl_t::sample_dx`

Horizontal radius for sample output [km].

Definition at line [878](#) of file [libtrac.h](#).

**4.3.2.138 sample\_dz** `double ctl_t::sample_dz`

Layer width for sample output [km].

Definition at line [881](#) of file [libtrac.h](#).

**4.3.2.139 stat\_basename** `char ctl_t::stat_basename[LEN]`

Basename of station data file.

Definition at line [884](#) of file [libtrac.h](#).

**4.3.2.140 stat\_lon** `double ctl_t::stat_lon`

Longitude of station [deg].

Definition at line [887](#) of file [libtrac.h](#).

**4.3.2.141 stat\_lat** `double ctl_t::stat_lat`

Latitude of station [deg].

Definition at line [890](#) of file [libtrac.h](#).

**4.3.2.142 stat\_r** double `ctl_t::stat_r`

Search radius around station [km].

Definition at line 893 of file [libtrac.h](#).

The documentation for this struct was generated from the following file:

- [libtrac.h](#)

**4.4 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].*
- float [ps](#) [EX][EY]  
*Surface pressure [hPa].*
- float [ts](#) [EX][EY]  
*Surface temperature [K].*
- float [zs](#) [EX][EY]  
*Surface geopotential height [km].*
- float [us](#) [EX][EY]  
*Surface zonal wind [m/s].*
- float [vs](#) [EX][EY]  
*Surface meridional wind [m/s].*
- float [pt](#) [EX][EY]  
*Tropopause pressure [hPa].*
- float [tt](#) [EX][EY]  
*Tropopause temperature [K].*
- float [zt](#) [EX][EY]  
*Tropopause geopotential height [km].*
- float [h2ot](#) [EX][EY]  
*Tropopause water vapor vmr [ppv].*



- float **pc** [EX][EY]  
*Cloud top pressure [hPa].*
- float **cl** [EX][EY]  
*Total column cloud water [kg/m<sup>2</sup>].*
- float **plcl** [EX][EY]  
*Pressure at lifted condensation level (LCL) [hPa].*
- float **plfc** [EX][EY]  
*Pressure at level of free convection (LFC) [hPa].*
- float **pel** [EX][EY]  
*Pressure at equilibrium level [hPa].*
- float **cape** [EX][EY]  
*Convective available potential energy [J/kg].*
- float **z** [EX][EY][EP]  
*Geopotential height at model levels [km].*
- 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 velocity [hPa/s].*
- float **pv** [EX][EY][EP]  
*Potential vorticity [PVU].*
- float **h2o** [EX][EY][EP]  
*Water vapor volume mixing ratio [1].*
- float **o3** [EX][EY][EP]  
*Ozone volume mixing ratio [1].*
- float **lwc** [EX][EY][EP]  
*Cloud liquid water content [kg/kg].*
- float **iwc** [EX][EY][EP]  
*Cloud ice water content [kg/kg].*
- float **pl** [EX][EY][EP]  
*Pressure on model levels [hPa].*

#### 4.4.1 Detailed Description

Meteorological data.

Definition at line 959 of file [libtrac.h](#).

#### 4.4.2 Field Documentation

##### 4.4.2.1 **time** double met\_t::time

Time [s].

Definition at line 962 of file [libtrac.h](#).

**4.4.2.2 nx** `int met_t::nx`

Number of longitudes.

Definition at line 965 of file [libtrac.h](#).

**4.4.2.3 ny** `int met_t::ny`

Number of latitudes.

Definition at line 968 of file [libtrac.h](#).

**4.4.2.4 np** `int met_t::np`

Number of pressure levels.

Definition at line 971 of file [libtrac.h](#).

**4.4.2.5 lon** `double met_t::lon[EX]`

Longitude [deg].

Definition at line 974 of file [libtrac.h](#).

**4.4.2.6 lat** `double met_t::lat[EY]`

Latitude [deg].

Definition at line 977 of file [libtrac.h](#).

**4.4.2.7 p** `double met_t::p[EP]`

Pressure [hPa].

Definition at line 980 of file [libtrac.h](#).

**4.4.2.8 ps** `float met_t::ps[EX][EY]`

Surface pressure [hPa].

Definition at line 983 of file [libtrac.h](#).

**4.4.2.9 ts** `float met_t::ts[EX][EY]`

Surface temperature [K].

Definition at line 986 of file [libtrac.h](#).

**4.4.2.10 zs** `float met_t::zs[EX][EY]`

Surface geopotential height [km].

Definition at line 989 of file [libtrac.h](#).

**4.4.2.11 us** `float met_t::us[EX][EY]`

Surface zonal wind [m/s].

Definition at line 992 of file [libtrac.h](#).

**4.4.2.12 vs** `float met_t::vs[EX][EY]`

Surface meridional wind [m/s].

Definition at line 995 of file [libtrac.h](#).

**4.4.2.13 pt** `float met_t::pt[EX][EY]`

Tropopause pressure [hPa].

Definition at line 998 of file [libtrac.h](#).

**4.4.2.14 tt** float met\_t::tt[EX][EY]

Tropopause temperature [K].

Definition at line 1001 of file [libtrac.h](#).

**4.4.2.15 zt** float met\_t::zt[EX][EY]

Tropopause geopotential height [km].

Definition at line 1004 of file [libtrac.h](#).

**4.4.2.16 h2ot** float met\_t::h2ot[EX][EY]

Tropopause water vapor vmr [ppv].

Definition at line 1007 of file [libtrac.h](#).

**4.4.2.17 pc** float met\_t::pc[EX][EY]

Cloud top pressure [hPa].

Definition at line 1010 of file [libtrac.h](#).

**4.4.2.18 cl** float met\_t::cl[EX][EY]

Total column cloud water [kg/m<sup>2</sup>].

Definition at line 1013 of file [libtrac.h](#).

**4.4.2.19 plcl** float met\_t::plcl[EX][EY]

Pressure at lifted condensation level (LCL) [hPa].

Definition at line 1016 of file [libtrac.h](#).

**4.4.2.20 plfc** `float met_t::plfc[EX][EY]`

Pressure at level of free convection (LFC) [hPa].

Definition at line [1019](#) of file [libtrac.h](#).

**4.4.2.21 pel** `float met_t::pel[EX][EY]`

Pressure at equilibrium level [hPa].

Definition at line [1022](#) of file [libtrac.h](#).

**4.4.2.22 cape** `float met_t::cape[EX][EY]`

Convective available potential energy [J/kg].

Definition at line [1025](#) of file [libtrac.h](#).

**4.4.2.23 z** `float met_t::z[EX][EY][EP]`

Geopotential height at model levels [km].

Definition at line [1028](#) of file [libtrac.h](#).

**4.4.2.24 t** `float met_t::t[EX][EY][EP]`

Temperature [K].

Definition at line [1031](#) of file [libtrac.h](#).

**4.4.2.25 u** `float met_t::u[EX][EY][EP]`

Zonal wind [m/s].

Definition at line [1034](#) of file [libtrac.h](#).

**4.4.2.26 v** float met\_t::v[EX][EY][EP]

Meridional wind [m/s].

Definition at line [1037](#) of file [libtrac.h](#).

**4.4.2.27 w** float met\_t::w[EX][EY][EP]

Vertical velocity [hPa/s].

Definition at line [1040](#) of file [libtrac.h](#).

**4.4.2.28 pv** float met\_t::pv[EX][EY][EP]

Potential vorticity [PVU].

Definition at line [1043](#) of file [libtrac.h](#).

**4.4.2.29 h2o** float met\_t::h2o[EX][EY][EP]

Water vapor volume mixing ratio [1].

Definition at line [1046](#) of file [libtrac.h](#).

**4.4.2.30 o3** float met\_t::o3[EX][EY][EP]

Ozone volume mixing ratio [1].

Definition at line [1049](#) of file [libtrac.h](#).

**4.4.2.31 lwc** float met\_t::lwc[EX][EY][EP]

Cloud liquid water content [kg/kg].

Definition at line [1052](#) of file [libtrac.h](#).

**4.4.2.32 iwc** `float met_t::iwc[EX][EY][EP]`

Cloud ice water content [kg/kg].

Definition at line [1055](#) of file [libtrac.h](#).

**4.4.2.33 pl** `float met_t::pl[EX][EY][EP]`

Pressure on model levels [hPa].

Definition at line [1058](#) of file [libtrac.h](#).

The documentation for this struct was generated from the following file:

- [libtrac.h](#)

## 5 File Documentation

### 5.1 atm\_conv.c File Reference

#### Functions

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

#### 5.1.1 Detailed Description

Convert file format of air parcel data files.

Definition in file [atm\\_conv.c](#).

#### 5.1.2 Function Documentation

```

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

```

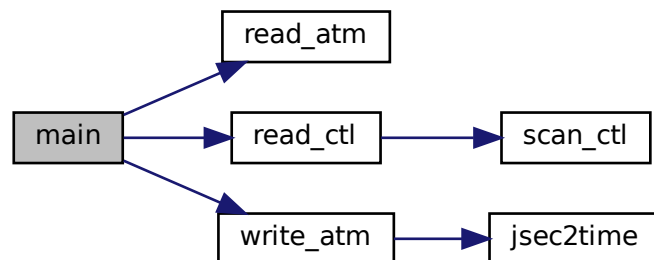
Definition at line 27 of file atm\_conv.c.

```

00029     {
00030
00031     ctl_t ctl;
00032
00033     atm_t *atm;
00034
00035     /* Check arguments... */
00036     if (argc < 6)
00037         ERRMSG("Give parameters: <ctl> <atm_in> <atm_in_type>"
00038              " <atm_out> <atm_out_type>");
00039
00040     /* Allocate... */
00041     ALLOC(atm, atm_t, 1);
00042
00043     /* Read control parameters... */
00044     read_ctl(argv[1], argc, argv, &ctl);
00045
00046     /* Read atmospheric data... */
00047     ctl.atm_type = atoi(argv[3]);
00048     if (!read_atm(argv[2], &ctl, atm))
00049         ERRMSG("Cannot open file!");
00050
00051     /* Write atmospheric data... */
00052     ctl.atm_type = atoi(argv[5]);
00053     write_atm(argv[4], &ctl, atm, 0);
00054
00055     /* Free... */
00056     free(atm);
00057
00058     return EXIT_SUCCESS;
00059 }

```

Here is the call graph for this function:



## 5.2 atm\_conv.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
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.

```



```

00016
00017 Copyright (C) 2013-2019 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     /* Check arguments... */
00036     if (argc < 6)
00037         ERRMSG("Give parameters: <ctl> <atm_in> <atm_in_type>"
00038             " <atm_out> <atm_out_type>");
00039
00040     /* Allocate... */
00041     ALLOC(atm, atm_t, 1);
00042
00043     /* Read control parameters... */
00044     read_ctl(argv[1], argc, argv, &ctl);
00045
00046     /* Read atmospheric data... */
00047     ctl.atm_type = atoi(argv[3]);
00048     if (!read_atm(argv[2], &ctl, atm))
00049         ERRMSG("Cannot open file!");
00050
00051     /* Write atmospheric data... */
00052     ctl.atm_type = atoi(argv[5]);
00053     write_atm(argv[4], &ctl, atm, 0);
00054
00055     /* Free... */
00056     free(atm);
00057
00058     return EXIT_SUCCESS;
00059 }

```

## 5.3 atm\_dist.c File Reference

### Functions

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

#### 5.3.1 Detailed Description

Calculate transport deviations of trajectories.

Definition in file [atm\\_dist.c](#).

#### 5.3.2 Function Documentation

```

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

```

Definition at line 27 of file [atm\\_dist.c](#).

```

00029     {
00030
00031         ctl_t ctl;
00032
00033         atm_t *atm1, *atm2;
00034
00035         FILE *out;
00036
00037         char tstr[LEN];
00038
00039         double *ahtd, *aqtd, *avtd, ahtdm, aqtdm[NQ], avtdm, lat0, lat1,
00040             *lat1_old, *lat2_old, *lh1, *lh2, lon0, lon1, *lon1_old, *lon2_old,
00041             *lv1, *lv2, p0, p1, *rhtd, *rqtd, *rvtd, rhtdm, rqtdm[NQ], rvtdm,
00042             t, t0 = 0, x0[3], x1[3], x2[3], z1, *z1_old, z2, *z2_old, *work;
00043
00044         int ens, f, init = 0, ip, iq, np, year, mon, day, hour, min;
00045
00046         /* Allocate... */
00047         ALLOC(atm1, atm_t, 1);
00048         ALLOC(atm2, atm_t, 1);
00049         ALLOC(lon1_old, double,
00050             NP);
00051         ALLOC(lat1_old, double,
00052             NP);
00053         ALLOC(z1_old, double,
00054             NP);
00055         ALLOC(lh1, double,
00056             NP);
00057         ALLOC(lv1, double,
00058             NP);
00059         ALLOC(lon2_old, double,
00060             NP);
00061         ALLOC(lat2_old, double,
00062             NP);
00063         ALLOC(z2_old, double,
00064             NP);
00065         ALLOC(lh2, double,
00066             NP);
00067         ALLOC(lv2, double,
00068             NP);
00069         ALLOC(ahtd, double,
00070             NP);
00071         ALLOC(avtd, double,
00072             NP);
00073         ALLOC(aqtd, double,
00074             NP * NQ);
00075         ALLOC(rhtd, double,
00076             NP);
00077         ALLOC(rvtd, double,
00078             NP);
00079         ALLOC(rqtd, double,
00080             NP * NQ);
00081         ALLOC(work, double,
00082             NP);
00083
00084         /* Check arguments... */
00085         if (argc < 6)
00086             ERRMSG("Give parameters: <ctl> <dist.tab> <param> <atmla> <atmlb>"
00087                 " [<atm2a> <atm2b> ...]");
00088
00089         /* Read control parameters... */
00090         read_ctl(argv[1], argc, argv, &ctl);
00091         ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-999", NULL);
00092         p0 = P(scan_ctl(argv[1], argc, argv, "DIST_Z0", -1, "-1000", NULL));
00093         p1 = P(scan_ctl(argv[1], argc, argv, "DIST_Z1", -1, "1000", NULL));
00094         lat0 = scan_ctl(argv[1], argc, argv, "DIST_LAT0", -1, "-1000", NULL);
00095         lat1 = scan_ctl(argv[1], argc, argv, "DIST_LAT1", -1, "1000", NULL);
00096         lon0 = scan_ctl(argv[1], argc, argv, "DIST_LON0", -1, "-1000", NULL);
00097         lon1 = scan_ctl(argv[1], argc, argv, "DIST_LON1", -1, "1000", NULL);
00098
00099         /* Write info... */
00100         printf("Write transport deviations: %s\n", argv[2]);
00101
00102         /* Create output file... */
00103         if (!(out = fopen(argv[2], "w")))
00104             ERRMSG("Cannot create file!");
00105
00106         /* Write header... */
00107         fprintf(out,
00108             "# $1 = time [s]\n"

```

```

00109         "# $2 = time difference [s]\n"
00110         "# $3 = absolute horizontal distance (%s) [km]\n"
00111         "# $4 = relative horizontal distance (%s) [%%]\n"
00112         "# $5 = absolute vertical distance (%s) [km]\n"
00113         "# $6 = relative vertical distance (%s) [%%]\n",
00114         argv[3], argv[3], argv[3], argv[3]);
00115     for (iq = 0; iq < ctl.nq; iq++)
00116         fprintf(out,
00117             "# $qd = %s absolute difference (%s) [%s]\n"
00118             "# $qd = %s relative difference (%s) [%%]\n",
00119             7 + 2 * iq, ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq],
00120             8 + 2 * iq, ctl.qnt_name[iq], argv[3]);
00121     fprintf(out, "# $qd = number of particles\n\n", 7 + 2 * ctl.nq);
00122
00123     /* Loop over file pairs... */
00124     for (f = 4; f < argc; f += 2) {
00125
00126         /* Read atmospheric data... */
00127         if (!read_atm(argv[f], &ctl, atm1) || !read_atm(argv[f + 1], &ctl, atm2))
00128             continue;
00129
00130         /* Check if structs match... */
00131         if (atm1->np != atm2->np)
00132             ERRMSG("Different numbers of particles!");
00133
00134         /* Get time from filename... */
00135         sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00136         year = atoi(tstr);
00137         sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00138         mon = atoi(tstr);
00139         sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00140         day = atoi(tstr);
00141         sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00142         hour = atoi(tstr);
00143         sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00144         min = atoi(tstr);
00145         time2jsec(year, mon, day, hour, min, 0, 0, &t);
00146
00147         /* Save initial time... */
00148         if (!init) {
00149             init = 1;
00150             t0 = t;
00151         }
00152
00153         /* Init... */
00154         np = 0;
00155         for (ip = 0; ip < atm1->np; ip++) {
00156             ahtd[ip] = avtd[ip] = rhtd[ip] = rvtd[ip] = 0;
00157             for (iq = 0; iq < ctl.nq; iq++)
00158                 aqtd[iq * NP + ip] = rqtd[iq * NP + ip] = 0;
00159         }
00160
00161         /* Loop over air parcels... */
00162         for (ip = 0; ip < atm1->np; ip++) {
00163
00164             /* Check data... */
00165             if (!gsl_finite(atm1->time[ip]) || !gsl_finite(atm2->time[ip]))
00166                 continue;
00167
00168             /* Check ensemble index... */
00169             if (ctl.qnt_ens > 0
00170                 && (atm1->q[ctl.qnt_ens][ip] != ens
00171                     || atm2->q[ctl.qnt_ens][ip] != ens))
00172                 continue;
00173
00174             /* Check spatial range... */
00175             if (atm1->p[ip] > p0 || atm1->p[ip] < p1
00176                 || atm1->lon[ip] < lon0 || atm1->lon[ip] > lon1
00177                 || atm1->lat[ip] < lat0 || atm1->lat[ip] > lat1)
00178                 continue;
00179             if (atm2->p[ip] > p0 || atm2->p[ip] < p1
00180                 || atm2->lon[ip] < lon0 || atm2->lon[ip] > lon1
00181                 || atm2->lat[ip] < lat0 || atm2->lat[ip] > lat1)
00182                 continue;
00183
00184             /* Convert coordinates... */
00185             geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00186             geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00187             z1 = Z(atm1->p[ip]);
00188             z2 = Z(atm2->p[ip]);
00189
00190             /* Calculate absolute transport deviations... */
00191             ahtd[np] = DIST(x1, x2);
00192             avtd[np] = z1 - z2;
00193             for (iq = 0; iq < ctl.nq; iq++)
00194                 aqtd[iq * NP + np] = atm1->q[iq][ip] - atm2->q[iq][ip];
00195

```

```

00196     /* Calculate relative transport deviations... */
00197     if (f > 4) {
00198
00199         /* Get trajectory lengths... */
00200         geo2cart(0, lon1_old[ip], lat1_old[ip], x0);
00201         lh1[ip] += DIST(x0, x1);
00202         lv1[ip] += fabs(z1_old[ip] - z1);
00203
00204         geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
00205         lh2[ip] += DIST(x0, x2);
00206         lv2[ip] += fabs(z2_old[ip] - z2);
00207
00208         /* Get relative transport deviations... */
00209         if (lh1[ip] + lh2[ip] > 0)
00210             rhtd[np] = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00211         if (lv1[ip] + lv2[ip] > 0)
00212             rvtd[np] = 200. * (z1 - z2) / (lv1[ip] + lv2[ip]);
00213     }
00214
00215     /* Get relative transport deviations... */
00216     for (iq = 0; iq < ctl.nq; iq++)
00217         rqtd[iq * NP + np] = 200. * (atm1->q[iq][ip] - atm2->q[iq][ip])
00218             / (fabs(atm1->q[iq][ip]) + fabs(atm2->q[iq][ip]));
00219
00220     /* Save positions of air parcels... */
00221     lon1_old[ip] = atm1->lon[ip];
00222     lat1_old[ip] = atm1->lat[ip];
00223     z1_old[ip] = z1;
00224
00225     lon2_old[ip] = atm2->lon[ip];
00226     lat2_old[ip] = atm2->lat[ip];
00227     z2_old[ip] = z2;
00228
00229     /* Increment air parcel counter... */
00230     np++;
00231 }
00232
00233 /* Get statistics... */
00234 if (strcasecmp(argv[3], "mean") == 0) {
00235     ahtdm = gsl_stats_mean(ahtd, 1, (size_t) np);
00236     rhtdm = gsl_stats_mean(rhtd, 1, (size_t) np);
00237     avtdm = gsl_stats_mean(avtd, 1, (size_t) np);
00238     rvtdm = gsl_stats_mean(rvtd, 1, (size_t) np);
00239     for (iq = 0; iq < ctl.nq; iq++) {
00240         aqtdm[iq] = gsl_stats_mean(&aqtd[iq * NP], 1, (size_t) np);
00241         rqtdm[iq] = gsl_stats_mean(&rqtd[iq * NP], 1, (size_t) np);
00242     }
00243 } else if (strcasecmp(argv[3], "stddev") == 0) {
00244     ahtdm = gsl_stats_sd(ahtd, 1, (size_t) np);
00245     rhtdm = gsl_stats_sd(rhtd, 1, (size_t) np);
00246     avtdm = gsl_stats_sd(avtd, 1, (size_t) np);
00247     rvtdm = gsl_stats_sd(rvtd, 1, (size_t) np);
00248     for (iq = 0; iq < ctl.nq; iq++) {
00249         aqtdm[iq] = gsl_stats_sd(&aqtd[iq * NP], 1, (size_t) np);
00250         rqtdm[iq] = gsl_stats_sd(&rqtd[iq * NP], 1, (size_t) np);
00251     }
00252 } else if (strcasecmp(argv[3], "min") == 0) {
00253     ahtdm = gsl_stats_min(ahtd, 1, (size_t) np);
00254     rhtdm = gsl_stats_min(rhtd, 1, (size_t) np);
00255     avtdm = gsl_stats_min(avtd, 1, (size_t) np);
00256     rvtdm = gsl_stats_min(rvtd, 1, (size_t) np);
00257     for (iq = 0; iq < ctl.nq; iq++) {
00258         aqtdm[iq] = gsl_stats_min(&aqtd[iq * NP], 1, (size_t) np);
00259         rqtdm[iq] = gsl_stats_min(&rqtd[iq * NP], 1, (size_t) np);
00260     }
00261 } else if (strcasecmp(argv[3], "max") == 0) {
00262     ahtdm = gsl_stats_max(ahtd, 1, (size_t) np);
00263     rhtdm = gsl_stats_max(rhtd, 1, (size_t) np);
00264     avtdm = gsl_stats_max(avtd, 1, (size_t) np);
00265     rvtdm = gsl_stats_max(rvtd, 1, (size_t) np);
00266     for (iq = 0; iq < ctl.nq; iq++) {
00267         aqtdm[iq] = gsl_stats_max(&aqtd[iq * NP], 1, (size_t) np);
00268         rqtdm[iq] = gsl_stats_max(&rqtd[iq * NP], 1, (size_t) np);
00269     }
00270 } else if (strcasecmp(argv[3], "skew") == 0) {
00271     ahtdm = gsl_stats_skew(ahtd, 1, (size_t) np);
00272     rhtdm = gsl_stats_skew(rhtd, 1, (size_t) np);
00273     avtdm = gsl_stats_skew(avtd, 1, (size_t) np);
00274     rvtdm = gsl_stats_skew(rvtd, 1, (size_t) np);
00275     for (iq = 0; iq < ctl.nq; iq++) {
00276         aqtdm[iq] = gsl_stats_skew(&aqtd[iq * NP], 1, (size_t) np);
00277         rqtdm[iq] = gsl_stats_skew(&rqtd[iq * NP], 1, (size_t) np);
00278     }
00279 } else if (strcasecmp(argv[3], "kurt") == 0) {
00280     ahtdm = gsl_stats_kurtosis(ahtd, 1, (size_t) np);
00281     rhtdm = gsl_stats_kurtosis(rhtd, 1, (size_t) np);
00282     avtdm = gsl_stats_kurtosis(avtd, 1, (size_t) np);

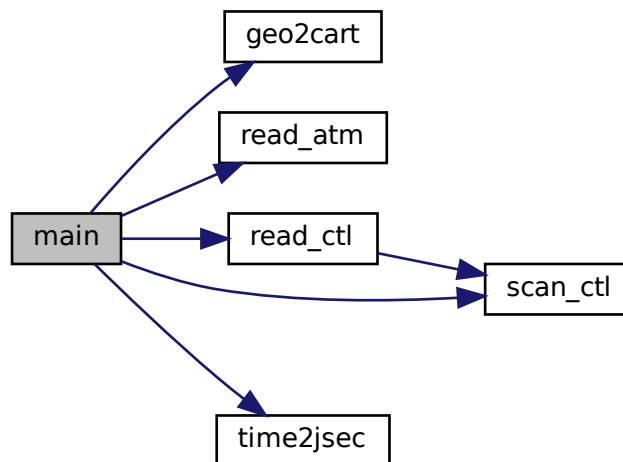
```

```

00283     rvtdm = gsl_stats_kurtosis(rvtd, 1, (size_t) np);
00284     for (iq = 0; iq < ctl.nq; iq++) {
00285         aqtdm[iq] = gsl_stats_kurtosis(&aqtd[iq * NP], 1, (size_t) np);
00286         rqtdm[iq] = gsl_stats_kurtosis(&rqtd[iq * NP], 1, (size_t) np);
00287     }
00288 } else if (strcasecmp(argv[3], "median") == 0) {
00289     ahtdm = gsl_stats_median(ahtd, 1, (size_t) np);
00290     rhtdm = gsl_stats_median(rhtd, 1, (size_t) np);
00291     avtdm = gsl_stats_median(avtd, 1, (size_t) np);
00292     rvtdm = gsl_stats_median(rvtd, 1, (size_t) np);
00293     for (iq = 0; iq < ctl.nq; iq++) {
00294         aqtdm[iq] = gsl_stats_median(&aqtd[iq * NP], 1, (size_t) np);
00295         rqtdm[iq] = gsl_stats_median(&rqtd[iq * NP], 1, (size_t) np);
00296     }
00297 } else if (strcasecmp(argv[3], "absdev") == 0) {
00298     ahtdm = gsl_stats_absdev(ahtd, 1, (size_t) np);
00299     rhtdm = gsl_stats_absdev(rhtd, 1, (size_t) np);
00300     avtdm = gsl_stats_absdev(avtd, 1, (size_t) np);
00301     rvtdm = gsl_stats_absdev(rvtd, 1, (size_t) np);
00302     for (iq = 0; iq < ctl.nq; iq++) {
00303         aqtdm[iq] = gsl_stats_absdev(&aqtd[iq * NP], 1, (size_t) np);
00304         rqtdm[iq] = gsl_stats_absdev(&rqtd[iq * NP], 1, (size_t) np);
00305     }
00306 } else if (strcasecmp(argv[3], "mad") == 0) {
00307     ahtdm = gsl_stats_mad0(ahtd, 1, (size_t) np, work);
00308     rhtdm = gsl_stats_mad0(rhtd, 1, (size_t) np, work);
00309     avtdm = gsl_stats_mad0(avtd, 1, (size_t) np, work);
00310     rvtdm = gsl_stats_mad0(rvtd, 1, (size_t) np, work);
00311     for (iq = 0; iq < ctl.nq; iq++) {
00312         aqtdm[iq] = gsl_stats_mad0(&aqtd[iq * NP], 1, (size_t) np, work);
00313         rqtdm[iq] = gsl_stats_mad0(&rqtd[iq * NP], 1, (size_t) np, work);
00314     }
00315 } else
00316     ERRMSG("Unknown parameter!");
00317
00318 /* Write output... */
00319 fprintf(out, "%.2f %.2f %g %g %g %g", t, t - t0,
00320         ahtdm, rhtdm, avtdm, rvtdm);
00321 for (iq = 0; iq < ctl.nq; iq++) {
00322     fprintf(out, " ");
00323     fprintf(out, ctl.qnt_format[iq], aqtdm[iq]);
00324     fprintf(out, " ");
00325     fprintf(out, ctl.qnt_format[iq], rqtdm[iq]);
00326 }
00327 fprintf(out, " %d\n", np);
00328 }
00329
00330 /* Close file... */
00331 fclose(out);
00332
00333 /* Free... */
00334 free(atm1);
00335 free(atm2);
00336 free(lon1_old);
00337 free(lat1_old);
00338 free(zl_old);
00339 free(lh1);
00340 free(lv1);
00341 free(lon2_old);
00342 free(lat2_old);
00343 free(z2_old);
00344 free(lh2);
00345 free(lv2);
00346 free(ahtd);
00347 free(avtd);
00348 free(aqtd);
00349 free(rhtd);
00350 free(rvtd);
00351 free(rqtd);
00352 free(work);
00353
00354 return EXIT_SUCCESS;
00355 }

```

Here is the call graph for this function:



## 5.4 atm\_dist.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
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018  */
00019
00020 #include "libtrac.h"
00021
00022 int main(
00023     int argc,
00024     char *argv[]) {
00025
00026     ctl_t ctl;
00027
00028     atm_t *atm1, *atm2;
00029
00030     FILE *out;
00031
00032     char tstr[LEN];
00033
00034     double *ahtd, *aqtd, *avtd, ahtdm, aqtdm[NQ], avtdm, lat0, lat1,
00035            *lat1_old, *lat2_old, *lh1, *lh2, lon0, lon1, *lon1_old, *lon2_old,
00036            *lv1, *lv2, p0, p1, *rhtd, *rqtd, *rvtd, rhtdm, rqtdm[NQ], rvtdm,
00037            t, t0 = 0, x0[3], x1[3], x2[3], z1, *z1_old, z2, *z2_old, *work;
00038
00039     int ens, f, init = 0, ip, iq, np, year, mon, day, hour, min;
00040
00041     /* Allocate... */
00042     ALLOC(atm1, atm_t, 1);
00043     ALLOC(atm2, atm_t, 1);
00044     ALLOC(lon1_old, double,
00045            NP);
00046     ALLOC(lat1_old, double,

```

```

00052     NP);
00053     ALLOC(z1_old, double,
00054     NP);
00055     ALLOC(lh1, double,
00056     NP);
00057     ALLOC(lv1, double,
00058     NP);
00059     ALLOC(lon2_old, double,
00060     NP);
00061     ALLOC(lat2_old, double,
00062     NP);
00063     ALLOC(z2_old, double,
00064     NP);
00065     ALLOC(lh2, double,
00066     NP);
00067     ALLOC(lv2, double,
00068     NP);
00069     ALLOC(ahtd, double,
00070     NP);
00071     ALLOC(avtd, double,
00072     NP);
00073     ALLOC(aqtd, double,
00074     NP * NQ);
00075     ALLOC(rhtd, double,
00076     NP);
00077     ALLOC(rvtd, double,
00078     NP);
00079     ALLOC(rqtd, double,
00080     NP * NQ);
00081     ALLOC(work, double,
00082     NP);
00083
00084     /* Check arguments... */
00085     if (argc < 6)
00086         ERRMSG("Give parameters: <ctl> <dist.tab> <param> <atmla> <atmlb>"
00087             " [<atm2a> <atm2b> ...]");
00088
00089     /* Read control parameters... */
00090     read_ctl(argv[1], argc, argv, &ctl);
00091     ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-999", NULL);
00092     p0 = P(scan_ctl(argv[1], argc, argv, "DIST_Z0", -1, "-1000", NULL));
00093     p1 = P(scan_ctl(argv[1], argc, argv, "DIST_Z1", -1, "1000", NULL));
00094     lat0 = scan_ctl(argv[1], argc, argv, "DIST_LAT0", -1, "-1000", NULL);
00095     lat1 = scan_ctl(argv[1], argc, argv, "DIST_LAT1", -1, "1000", NULL);
00096     lon0 = scan_ctl(argv[1], argc, argv, "DIST_LON0", -1, "-1000", NULL);
00097     lon1 = scan_ctl(argv[1], argc, argv, "DIST_LON1", -1, "1000", NULL);
00098
00099     /* Write info... */
00100     printf("Write transport deviations: %s\n", argv[2]);
00101
00102     /* Create output file... */
00103     if (!(out = fopen(argv[2], "w")))
00104         ERRMSG("Cannot create file!");
00105
00106     /* Write header... */
00107     fprintf(out,
00108         "# $1 = time [s]\n"
00109         "# $2 = time difference [s]\n"
00110         "# $3 = absolute horizontal distance (%s) [km]\n"
00111         "# $4 = relative horizontal distance (%s) [%%]\n"
00112         "# $5 = absolute vertical distance (%s) [km]\n"
00113         "# $6 = relative vertical distance (%s) [%%]\n",
00114         argv[3], argv[3], argv[3], argv[3]);
00115     for (iq = 0; iq < ctl.nq; iq++)
00116         fprintf(out,
00117             "# $%d = %s absolute difference (%s) [%s]\n"
00118             "# $%d = %s relative difference (%s) [%%]\n",
00119             7 + 2 * iq, ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq],
00120             8 + 2 * iq, ctl.qnt_name[iq], argv[3]);
00121     fprintf(out, "# $%d = number of particles\n\n", 7 + 2 * ctl.nq);
00122
00123     /* Loop over file pairs... */
00124     for (f = 4; f < argc; f += 2) {
00125
00126         /* Read atmospheric data... */
00127         if (!read_atm(argv[f], &ctl, atm1) || !read_atm(argv[f + 1], &ctl, atm2))
00128             continue;
00129
00130         /* Check if structs match... */
00131         if (atm1->np != atm2->np)
00132             ERRMSG("Different numbers of particles!");
00133
00134         /* Get time from filename... */
00135         sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00136         year = atoi(tstr);
00137         sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00138         mon = atoi(tstr);

```

```

00139     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00140     day = atoi(tstr);
00141     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00142     hour = atoi(tstr);
00143     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00144     min = atoi(tstr);
00145     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00146
00147     /* Save initial time... */
00148     if (!init) {
00149         init = 1;
00150         t0 = t;
00151     }
00152
00153     /* Init... */
00154     np = 0;
00155     for (ip = 0; ip < atm1->np; ip++) {
00156         ahtd[ip] = avtd[ip] = rhtd[ip] = rvtd[ip] = 0;
00157         for (iq = 0; iq < ctl.nq; iq++)
00158             aqtd[iq * NP + ip] = rqtd[iq * NP + ip] = 0;
00159     }
00160
00161     /* Loop over air parcels... */
00162     for (ip = 0; ip < atm1->np; ip++) {
00163
00164         /* Check data... */
00165         if (!gsl_finite(atm1->time[ip]) || !gsl_finite(atm2->time[ip]))
00166             continue;
00167
00168         /* Check ensemble index... */
00169         if (ctl.qnt_ens > 0
00170             && (atm1->q[ctl.qnt_ens][ip] != ens
00171                 || atm2->q[ctl.qnt_ens][ip] != ens))
00172             continue;
00173
00174         /* Check spatial range... */
00175         if (atm1->p[ip] > p0 || atm1->p[ip] < p1
00176             || atm1->lon[ip] < lon0 || atm1->lon[ip] > lon1
00177             || atm1->lat[ip] < lat0 || atm1->lat[ip] > lat1)
00178             continue;
00179         if (atm2->p[ip] > p0 || atm2->p[ip] < p1
00180             || atm2->lon[ip] < lon0 || atm2->lon[ip] > lon1
00181             || atm2->lat[ip] < lat0 || atm2->lat[ip] > lat1)
00182             continue;
00183
00184         /* Convert coordinates... */
00185         geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00186         geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00187         z1 = Z(atm1->p[ip]);
00188         z2 = Z(atm2->p[ip]);
00189
00190         /* Calculate absolute transport deviations... */
00191         ahtd[np] = DIST(x1, x2);
00192         avtd[np] = z1 - z2;
00193         for (iq = 0; iq < ctl.nq; iq++)
00194             aqtd[iq * NP + np] = atm1->q[iq][ip] - atm2->q[iq][ip];
00195
00196         /* Calculate relative transport deviations... */
00197         if (f > 4) {
00198
00199             /* Get trajectory lengths... */
00200             geo2cart(0, lon1_old[ip], lat1_old[ip], x0);
00201             lh1[ip] += DIST(x0, x1);
00202             lv1[ip] += fabs(z1_old[ip] - z1);
00203
00204             geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
00205             lh2[ip] += DIST(x0, x2);
00206             lv2[ip] += fabs(z2_old[ip] - z2);
00207
00208             /* Get relative transport deviations... */
00209             if (lh1[ip] + lh2[ip] > 0)
00210                 rhtd[np] = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00211             if (lv1[ip] + lv2[ip] > 0)
00212                 rvtd[np] = 200. * (z1 - z2) / (lv1[ip] + lv2[ip]);
00213         }
00214
00215         /* Get relative transport deviations... */
00216         for (iq = 0; iq < ctl.nq; iq++)
00217             rqtd[iq * NP + np] = 200. * (atm1->q[iq][ip] - atm2->q[iq][ip])
00218                 / (fabs(atm1->q[iq][ip]) + fabs(atm2->q[iq][ip]));
00219
00220         /* Save positions of air parcels... */
00221         lon1_old[ip] = atm1->lon[ip];
00222         lat1_old[ip] = atm1->lat[ip];
00223         z1_old[ip] = z1;
00224
00225         lon2_old[ip] = atm2->lon[ip];

```



```

00226     lat2_old[ip] = atm2->lat[ip];
00227     z2_old[ip] = z2;
00228
00229     /* Increment air parcel counter... */
00230     np++;
00231 }
00232
00233 /* Get statistics... */
00234 if (strcasecmp(argv[3], "mean") == 0) {
00235     ahtdm = gsl_stats_mean(ahtd, 1, (size_t) np);
00236     rhtdm = gsl_stats_mean(rhtd, 1, (size_t) np);
00237     avtdm = gsl_stats_mean(avtd, 1, (size_t) np);
00238     rvtdm = gsl_stats_mean(rvtd, 1, (size_t) np);
00239     for (iq = 0; iq < ctl.nq; iq++) {
00240         aqtdm[iq] = gsl_stats_mean(&aqtd[iq * NP], 1, (size_t) np);
00241         rqtdm[iq] = gsl_stats_mean(&rqtd[iq * NP], 1, (size_t) np);
00242     }
00243 } else if (strcasecmp(argv[3], "stddev") == 0) {
00244     ahtdm = gsl_stats_sd(ahtd, 1, (size_t) np);
00245     rhtdm = gsl_stats_sd(rhtd, 1, (size_t) np);
00246     avtdm = gsl_stats_sd(avtd, 1, (size_t) np);
00247     rvtdm = gsl_stats_sd(rvtd, 1, (size_t) np);
00248     for (iq = 0; iq < ctl.nq; iq++) {
00249         aqtdm[iq] = gsl_stats_sd(&aqtd[iq * NP], 1, (size_t) np);
00250         rqtdm[iq] = gsl_stats_sd(&rqtd[iq * NP], 1, (size_t) np);
00251     }
00252 } else if (strcasecmp(argv[3], "min") == 0) {
00253     ahtdm = gsl_stats_min(ahtd, 1, (size_t) np);
00254     rhtdm = gsl_stats_min(rhtd, 1, (size_t) np);
00255     avtdm = gsl_stats_min(avtd, 1, (size_t) np);
00256     rvtdm = gsl_stats_min(rvtd, 1, (size_t) np);
00257     for (iq = 0; iq < ctl.nq; iq++) {
00258         aqtdm[iq] = gsl_stats_min(&aqtd[iq * NP], 1, (size_t) np);
00259         rqtdm[iq] = gsl_stats_min(&rqtd[iq * NP], 1, (size_t) np);
00260     }
00261 } else if (strcasecmp(argv[3], "max") == 0) {
00262     ahtdm = gsl_stats_max(ahtd, 1, (size_t) np);
00263     rhtdm = gsl_stats_max(rhtd, 1, (size_t) np);
00264     avtdm = gsl_stats_max(avtd, 1, (size_t) np);
00265     rvtdm = gsl_stats_max(rvtd, 1, (size_t) np);
00266     for (iq = 0; iq < ctl.nq; iq++) {
00267         aqtdm[iq] = gsl_stats_max(&aqtd[iq * NP], 1, (size_t) np);
00268         rqtdm[iq] = gsl_stats_max(&rqtd[iq * NP], 1, (size_t) np);
00269     }
00270 } else if (strcasecmp(argv[3], "skew") == 0) {
00271     ahtdm = gsl_stats_skew(ahtd, 1, (size_t) np);
00272     rhtdm = gsl_stats_skew(rhtd, 1, (size_t) np);
00273     avtdm = gsl_stats_skew(avtd, 1, (size_t) np);
00274     rvtdm = gsl_stats_skew(rvtd, 1, (size_t) np);
00275     for (iq = 0; iq < ctl.nq; iq++) {
00276         aqtdm[iq] = gsl_stats_skew(&aqtd[iq * NP], 1, (size_t) np);
00277         rqtdm[iq] = gsl_stats_skew(&rqtd[iq * NP], 1, (size_t) np);
00278     }
00279 } else if (strcasecmp(argv[3], "kurt") == 0) {
00280     ahtdm = gsl_stats_kurtosis(ahtd, 1, (size_t) np);
00281     rhtdm = gsl_stats_kurtosis(rhtd, 1, (size_t) np);
00282     avtdm = gsl_stats_kurtosis(avtd, 1, (size_t) np);
00283     rvtdm = gsl_stats_kurtosis(rvtd, 1, (size_t) np);
00284     for (iq = 0; iq < ctl.nq; iq++) {
00285         aqtdm[iq] = gsl_stats_kurtosis(&aqtd[iq * NP], 1, (size_t) np);
00286         rqtdm[iq] = gsl_stats_kurtosis(&rqtd[iq * NP], 1, (size_t) np);
00287     }
00288 } else if (strcasecmp(argv[3], "median") == 0) {
00289     ahtdm = gsl_stats_median(ahtd, 1, (size_t) np);
00290     rhtdm = gsl_stats_median(rhtd, 1, (size_t) np);
00291     avtdm = gsl_stats_median(avtd, 1, (size_t) np);
00292     rvtdm = gsl_stats_median(rvtd, 1, (size_t) np);
00293     for (iq = 0; iq < ctl.nq; iq++) {
00294         aqtdm[iq] = gsl_stats_median(&aqtd[iq * NP], 1, (size_t) np);
00295         rqtdm[iq] = gsl_stats_median(&rqtd[iq * NP], 1, (size_t) np);
00296     }
00297 } else if (strcasecmp(argv[3], "absdev") == 0) {
00298     ahtdm = gsl_stats_absdev(ahtd, 1, (size_t) np);
00299     rhtdm = gsl_stats_absdev(rhtd, 1, (size_t) np);
00300     avtdm = gsl_stats_absdev(avtd, 1, (size_t) np);
00301     rvtdm = gsl_stats_absdev(rvtd, 1, (size_t) np);
00302     for (iq = 0; iq < ctl.nq; iq++) {
00303         aqtdm[iq] = gsl_stats_absdev(&aqtd[iq * NP], 1, (size_t) np);
00304         rqtdm[iq] = gsl_stats_absdev(&rqtd[iq * NP], 1, (size_t) np);
00305     }
00306 } else if (strcasecmp(argv[3], "mad") == 0) {
00307     ahtdm = gsl_stats_mad0(ahtd, 1, (size_t) np, work);
00308     rhtdm = gsl_stats_mad0(rhtd, 1, (size_t) np, work);
00309     avtdm = gsl_stats_mad0(avtd, 1, (size_t) np, work);
00310     rvtdm = gsl_stats_mad0(rvtd, 1, (size_t) np, work);
00311     for (iq = 0; iq < ctl.nq; iq++) {
00312         aqtdm[iq] = gsl_stats_mad0(&aqtd[iq * NP], 1, (size_t) np, work);

```

```

00313         rqtdm[iq] = gsl_stats_mad0(&rqtd[iq * NP], 1, (size_t) np, work);
00314     }
00315 } else
00316     ERRMSG("Unknown parameter!");
00317
00318 /* Write output... */
00319 fprintf(out, "%.2f %.2f %g %g %g %g", t, t - t0,
00320         ahtdm, rhtdm, avtdm, rvtdm);
00321 for (iq = 0; iq < ctl.nq; iq++) {
00322     fprintf(out, " ");
00323     fprintf(out, ctl.qnt_format[iq], aqtdm[iq]);
00324     fprintf(out, " ");
00325     fprintf(out, ctl.qnt_format[iq], rqtdm[iq]);
00326 }
00327 fprintf(out, " %d\n", np);
00328 }
00329
00330 /* Close file... */
00331 fclose(out);
00332
00333 /* Free... */
00334 free(atm1);
00335 free(atm2);
00336 free(lon1_old);
00337 free(lat1_old);
00338 free(z1_old);
00339 free(lh1);
00340 free(lv1);
00341 free(lon2_old);
00342 free(lat2_old);
00343 free(z2_old);
00344 free(lh2);
00345 free(lv2);
00346 free(ahtd);
00347 free(avtd);
00348 free(aqtd);
00349 free(rhtd);
00350 free(rvtd);
00351 free(rqtd);
00352 free(work);
00353
00354 return EXIT_SUCCESS;
00355 }

```

## 5.5 atm\_init.c File Reference

### Functions

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

#### 5.5.1 Detailed Description

Create atmospheric data file with initial air parcel positions.

Definition in file [atm\\_init.c](#).

#### 5.5.2 Function Documentation

```

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

```

Definition at line 27 of file [atm\\_init.c](#).

```

00029     {
00030
00031         atm_t *atm;
00032
00033         ctl_t ctl;
00034
00035         gsl_rng *rng;
00036
00037         double dt, dz, dlon, dlat, lat0, lat1, lon0, lon1, t0, t1, z0, z1,
00038             t, z, lon, lat, st, sz, slon, slat, sx, ut, uz, ulon, ulat, m;
00039
00040         int 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... */
00050         read_ctl(argv[1], argc, argv, &ctl);
00051         t0 = scan_ctl(argv[1], argc, argv, "INIT_T0", -1, "0", NULL);
00052         t1 = scan_ctl(argv[1], argc, argv, "INIT_T1", -1, "0", NULL);
00053         dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
00054         z0 = scan_ctl(argv[1], argc, argv, "INIT_Z0", -1, "0", NULL);
00055         z1 = scan_ctl(argv[1], argc, argv, "INIT_Z1", -1, "0", NULL);
00056         dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);
00057         lon0 = scan_ctl(argv[1], argc, argv, "INIT_LON0", -1, "0", NULL);
00058         lon1 = scan_ctl(argv[1], argc, argv, "INIT_LON1", -1, "0", NULL);
00059         dlon = scan_ctl(argv[1], argc, argv, "INIT_DLON", -1, "1", NULL);
00060         lat0 = scan_ctl(argv[1], argc, argv, "INIT_LAT0", -1, "0", NULL);
00061         lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT1", -1, "0", NULL);
00062         dlat = scan_ctl(argv[1], argc, argv, "INIT_DLAT", -1, "1", NULL);
00063         st = scan_ctl(argv[1], argc, argv, "INIT_ST", -1, "0", NULL);
00064         sz = scan_ctl(argv[1], argc, argv, "INIT_SZ", -1, "0", NULL);
00065         slon = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
00066         slat = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
00067         sx = scan_ctl(argv[1], argc, argv, "INIT_SX", -1, "0", NULL);
00068         ut = scan_ctl(argv[1], argc, argv, "INIT_UT", -1, "0", NULL);
00069         uz = scan_ctl(argv[1], argc, argv, "INIT_UZ", -1, "0", NULL);
00070         ulon = scan_ctl(argv[1], argc, argv, "INIT_ULON", -1, "0", NULL);
00071         ulat = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
00072         even = (int) scan_ctl(argv[1], argc, argv, "INIT_EVENLY", -1, "0", NULL);
00073         rep = (int) scan_ctl(argv[1], argc, argv, "INIT_REP", -1, "1", NULL);
00074         m = scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
00075
00076         /* Initialize random number generator... */
00077         gsl_rng_env_setup();
00078         rng = gsl_rng_alloc(gsl_rng_default);
00079
00080         /* Create grid... */
00081         for (t = t0; t <= t1; t += dt)
00082             for (z = z0; z <= z1; z += dz)
00083                 for (lon = lon0; lon <= lon1; lon += dlon)
00084                     for (lat = lat0; lat <= lat1; lat += dlat)
00085                         for (irep = 0; irep < rep; irep++) {
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]
00095                                 = (lon + gsl_ran_gaussian_ziggurat(rng, slon / 2.3548)
00096                                    + gsl_ran_gaussian_ziggurat(rng, DX2DEG(sx, lat) / 2.3548)
00097                                    + ulon * (gsl_rng_uniform(rng) - 0.5));
00098                             do {
00099                                 atm->lat[atm->np]
00100                                     = (lat + gsl_ran_gaussian_ziggurat(rng, slat / 2.3548)
00101                                        + gsl_ran_gaussian_ziggurat(rng, DY2DEG(sx) / 2.3548)
00102                                        + ulat * (gsl_rng_uniform(rng) - 0.5));
00103                             } while (even && gsl_rng_uniform(rng) >
00104                                     fabs(cos(atm->lat[atm->np] * M_PI / 180.)));
00105
00106                             /* Set particle counter... */
00107                             if ((++atm->np) > NP)
00108                                 ERRMSG("Too many particles!");
00109                         }
00110             }
00111     }

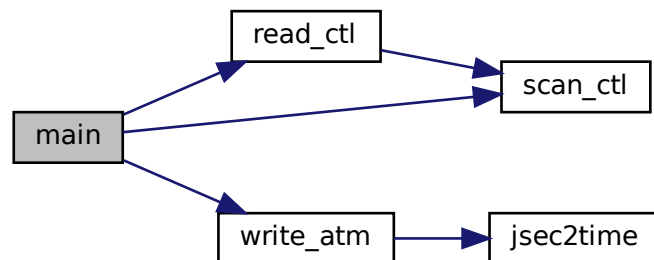
```

```

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... */
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, 0);
00122
00123     /* Free... */
00124     gsl_rng_free(rng);
00125     free(atm);
00126
00127     return EXIT_SUCCESS;
00128 }

```

Here is the call graph for this function:



## 5.6 atm\_init.c

```

00001 /*
00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.
00008
00009  MPTRAC is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU General Public License for more details.
00013
00014  You should have received a copy of the GNU General Public License
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028     int argc,
00029     char *argv[]) {
00030
00031     atm_t *atm;
00032
00033     ctl_t ctl;
00034
00035     gsl_rng *rng;
00036
00037     double dt, dz, dlon, dlat, lat0, lat1, lon0, lon1, t0, t1, z0, z1,
00038            t, z, lon, lat, st, sz, slon, slat, sx, ut, uz, ulon, ulat, m;

```

```

00039
00040     int 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... */
00050     read_ctl(argv[1], argc, argv, &ctl);
00051     t0 = scan_ctl(argv[1], argc, argv, "INIT_T0", -1, "0", NULL);
00052     t1 = scan_ctl(argv[1], argc, argv, "INIT_T1", -1, "0", NULL);
00053     dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
00054     z0 = scan_ctl(argv[1], argc, argv, "INIT_Z0", -1, "0", NULL);
00055     z1 = scan_ctl(argv[1], argc, argv, "INIT_Z1", -1, "0", NULL);
00056     dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);
00057     lon0 = scan_ctl(argv[1], argc, argv, "INIT_LON0", -1, "0", NULL);
00058     lon1 = scan_ctl(argv[1], argc, argv, "INIT_LON1", -1, "0", NULL);
00059     dlon = scan_ctl(argv[1], argc, argv, "INIT_DLON", -1, "1", NULL);
00060     lat0 = scan_ctl(argv[1], argc, argv, "INIT_LAT0", -1, "0", NULL);
00061     lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT1", -1, "0", NULL);
00062     dlat = scan_ctl(argv[1], argc, argv, "INIT_PLAT", -1, "1", NULL);
00063     st = scan_ctl(argv[1], argc, argv, "INIT_ST", -1, "0", NULL);
00064     sz = scan_ctl(argv[1], argc, argv, "INIT_SZ", -1, "0", NULL);
00065     slon = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
00066     slat = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
00067     sx = scan_ctl(argv[1], argc, argv, "INIT_SX", -1, "0", NULL);
00068     ut = scan_ctl(argv[1], argc, argv, "INIT_UT", -1, "0", NULL);
00069     uz = scan_ctl(argv[1], argc, argv, "INIT_UZ", -1, "0", NULL);
00070     ulon = scan_ctl(argv[1], argc, argv, "INIT_ULON", -1, "0", NULL);
00071     ulat = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
00072     even = (int) scan_ctl(argv[1], argc, argv, "INIT_EVENLY", -1, "0", NULL);
00073     rep = (int) scan_ctl(argv[1], argc, argv, "INIT_REP", -1, "1", NULL);
00074     m = scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
00075
00076     /* Initialize random number generator... */
00077     gsl_rng_env_setup();
00078     rng = gsl_rng_alloc(gsl_rng_default);
00079
00080     /* Create grid... */
00081     for (t = t0; t <= t1; t += dt)
00082         for (z = z0; z <= z1; z += dz)
00083             for (lon = lon0; lon <= lon1; lon += dlon)
00084                 for (lat = lat0; lat <= lat1; lat += dlat)
00085                     for (irep = 0; irep < rep; irep++) {
00086
00087                         /* Set position... */
00088                         atm->time[atm->np]
00089                             = (t + gsl_rng_gaussian_ziggurat(rng, st / 2.3548)
00090                                + ut * (gsl_rng_uniform(rng) - 0.5));
00091                         atm->p[atm->np]
00092                             = P(z + gsl_rng_gaussian_ziggurat(rng, sz / 2.3548)
00093                                + uz * (gsl_rng_uniform(rng) - 0.5));
00094                         atm->lon[atm->np]
00095                             = (lon + gsl_rng_gaussian_ziggurat(rng, slon / 2.3548)
00096                                + gsl_rng_gaussian_ziggurat(rng, DX2DEG(sx, lat) / 2.3548)
00097                                + ulon * (gsl_rng_uniform(rng) - 0.5));
00098                         do {
00099                             atm->lat[atm->np]
00100                                 = (lat + gsl_rng_gaussian_ziggurat(rng, slat / 2.3548)
00101                                    + gsl_rng_gaussian_ziggurat(rng, DY2DEG(sx) / 2.3548)
00102                                    + ulat * (gsl_rng_uniform(rng) - 0.5));
00103                         } while (even && gsl_rng_uniform(rng) >
00104                                fabs(cos(atm->lat[atm->np] * M_PI / 180.)));
00105
00106                         /* Set particle counter... */
00107                         if ((++atm->np) > NP)
00108                             ERRMSG("Too many particles!");
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... */
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, 0);
00122
00123     /* Free... */
00124     gsl_rng_free(rng);
00125     free(atm);

```

```

00126
00127     return EXIT_SUCCESS;
00128 }

```

## 5.7 atm\_select.c File Reference

### Functions

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

#### 5.7.1 Detailed Description

Extract subsets of air parcels from atmospheric data files.

Definition in file [atm\\_select.c](#).

#### 5.7.2 Function Documentation

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

Definition at line 27 of file [atm\\_select.c](#).

```

00029     {
00030
00031         ctl_t ctl;
00032
00033         atm_t *atm, *atm2;
00034
00035         double lat0, lat1, lon0, lon1, p0, p1, r, r0, r1, rlon, rlat, t0, t1, x0[3],
00036             x1[3];
00037
00038         int f, ip, ip0, ip1, iq, stride;
00039
00040         /* Allocate... */
00041         ALLOC(atm, atm_t, 1);
00042         ALLOC(atm2, atm_t, 1);
00043
00044         /* Check arguments... */
00045         if (argc < 4)
00046             ERRMSG("Give parameters: <ctl> <atm_select> <atm1> [<atm2> ...]");
00047
00048         /* Read control parameters... */
00049         read_ctl(argv[1], argc, argv, &ctl);
00050         stride =
00051             (int) scan_ctl(argv[1], argc, argv, "SELECT_STRIDE", -1, "1", NULL);
00052         ip0 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP0", -1, "-999", NULL);
00053         ip1 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP1", -1, "-999", NULL);
00054         t0 = scan_ctl(argv[1], argc, argv, "SELECT_T0", -1, "0", NULL);
00055         t1 = scan_ctl(argv[1], argc, argv, "SELECT_T1", -1, "0", NULL);
00056         p0 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z0", -1, "0", NULL));
00057         p1 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z1", -1, "0", NULL));
00058         lon0 = scan_ctl(argv[1], argc, argv, "SELECT_LON0", -1, "0", NULL);
00059         lon1 = scan_ctl(argv[1], argc, argv, "SELECT_LON1", -1, "0", NULL);
00060         lat0 = scan_ctl(argv[1], argc, argv, "SELECT_LAT0", -1, "0", NULL);
00061         lat1 = scan_ctl(argv[1], argc, argv, "SELECT_LAT1", -1, "0", NULL);
00062         r0 = scan_ctl(argv[1], argc, argv, "SELECT_R0", -1, "0", NULL);
00063         r1 = scan_ctl(argv[1], argc, argv, "SELECT_R1", -1, "0", NULL);
00064         rlon = scan_ctl(argv[1], argc, argv, "SELECT_RLON", -1, "0", NULL);
00065         rlat = scan_ctl(argv[1], argc, argv, "SELECT_RLAT", -1, "0", NULL);
00066
00067         /* Get Cartesian coordinates... */
00068         geo2cart(0, rlon, rlat, x0);
00069

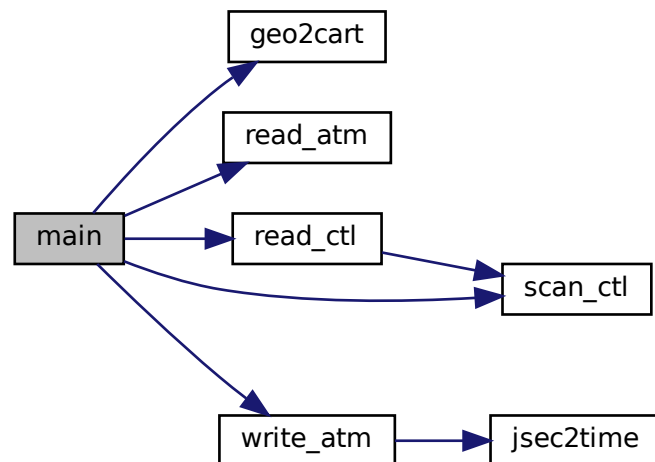
```

```

00070  /* Loop over files... */
00071  for (f = 3; f < argc; f++) {
00072
00073      /* Read atmospheric data... */
00074      if (!read_atm(argv[f], &ctl, atm))
00075          continue;
00076
00077      /* Adjust range of air parcels... */
00078      if (ip0 < 0)
00079          ip0 = 0;
00080      ip0 = GSL_MIN(ip0, atm->np - 1);
00081      if (ip1 < 0)
00082          ip1 = atm->np - 1;
00083      ip1 = GSL_MIN(ip1, atm->np - 1);
00084      if (ip1 < ip0)
00085          ip1 = ip0;
00086
00087      /* Loop over air parcels... */
00088      for (ip = ip0; ip <= ip1; ip += stride) {
00089
00090          /* Check time... */
00091          if (t0 != t1)
00092              if ((t1 > t0 && (atm->time[ip] < t0 || atm->time[ip] > t1))
00093                  || (t1 < t0 && (atm->time[ip] < t0 && atm->time[ip] > t1)))
00094                  continue;
00095
00096          /* Check vertical distance... */
00097          if (p0 != p1)
00098              if ((p0 > p1 && (atm->p[ip] > p0 || atm->p[ip] < p1))
00099                  || (p0 < p1 && (atm->p[ip] > p0 && atm->p[ip] < p1)))
00100                  continue;
00101
00102          /* Check longitude... */
00103          if (lon0 != lon1)
00104              if ((lon1 > lon0 && (atm->lon[ip] < lon0 || atm->lon[ip] > lon1))
00105                  || (lon1 < lon0 && (atm->lon[ip] < lon0 && atm->lon[ip] > lon1)))
00106                  continue;
00107
00108          /* Check latitude... */
00109          if (lat0 != lat1)
00110              if ((lat1 > lat0 && (atm->lat[ip] < lat0 || atm->lat[ip] > lat1))
00111                  || (lat1 < lat0 && (atm->lat[ip] < lat0 && atm->lat[ip] > lat1)))
00112                  continue;
00113
00114          /* Check horizontal distance... */
00115          if (r0 != r1) {
00116              geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
00117              r = DIST(x0, x1);
00118              if ((r1 > r0 && (r < r0 || r > r1))
00119                  || (r1 < r0 && (r < r0 && r > r1)))
00120                  continue;
00121          }
00122
00123          /* Copy data... */
00124          atm2->time[atm2->np] = atm->time[ip];
00125          atm2->p[atm2->np] = atm->p[ip];
00126          atm2->lon[atm2->np] = atm->lon[ip];
00127          atm2->lat[atm2->np] = atm->lat[ip];
00128          for (iq = 0; iq < ctl.nq; iq++)
00129              atm2->q[iq][atm2->np] = atm->q[iq][ip];
00130          if ((++atm2->np) > NP)
00131              ERRMSG("Too many air parcels!");
00132      }
00133  }
00134
00135  /* Close file... */
00136  write_atm(argv[2], &ctl, atm2, 0);
00137
00138  /* Free... */
00139  free(atm);
00140  free(atm2);
00141
00142  return EXIT_SUCCESS;
00143 }

```

Here is the call graph for this function:



## 5.8 atm\_select.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
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2021 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, *atm2;
00034
00035     double lat0, lat1, lon0, lon1, p0, p1, r, r0, r1, rlon, rlat, t0, t1, x0[3],
00036            x1[3];
00037
00038     int f, ip, ip0, ip1, iq, stride;
00039
00040     /* Allocate... */
00041     ALLOC(atm, atm_t, 1);
00042     ALLOC(atm2, atm_t, 1);
00043
00044     /* Check arguments... */
00045     if (argc < 4)
00046         ERRMSG("Give parameters: <ctl> <atm_select> <atm1> [<atm2> ...]");
00047
00048     /* Read control parameters... */
00049     read_ctl(argv[1], argc, argv, &ctl);
00050     stride =
00051         (int) scan_ctl(argv[1], argc, argv, "SELECT_STRIDE", -1, "1", NULL);

```



```

00052 ip0 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP0", -1, "-999", NULL);
00053 ip1 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP1", -1, "-999", NULL);
00054 t0 = scan_ctl(argv[1], argc, argv, "SELECT_T0", -1, "0", NULL);
00055 t1 = scan_ctl(argv[1], argc, argv, "SELECT_T1", -1, "0", NULL);
00056 p0 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z0", -1, "0", NULL));
00057 p1 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z1", -1, "0", NULL));
00058 lon0 = scan_ctl(argv[1], argc, argv, "SELECT_LON0", -1, "0", NULL);
00059 lon1 = scan_ctl(argv[1], argc, argv, "SELECT_LON1", -1, "0", NULL);
00060 lat0 = scan_ctl(argv[1], argc, argv, "SELECT_LAT0", -1, "0", NULL);
00061 lat1 = scan_ctl(argv[1], argc, argv, "SELECT_LAT1", -1, "0", NULL);
00062 r0 = scan_ctl(argv[1], argc, argv, "SELECT_R0", -1, "0", NULL);
00063 r1 = scan_ctl(argv[1], argc, argv, "SELECT_R1", -1, "0", NULL);
00064 rlon = scan_ctl(argv[1], argc, argv, "SELECT_RLON", -1, "0", NULL);
00065 rlat = scan_ctl(argv[1], argc, argv, "SELECT_RLAT", -1, "0", NULL);
00066
00067 /* Get Cartesian coordinates... */
00068 geo2cart(0, rlon, rlat, x0);
00069
00070 /* Loop over files... */
00071 for (f = 3; f < argc; f++) {
00072
00073     /* Read atmospheric data... */
00074     if (!read_atm(argv[f], &ctl, atm))
00075         continue;
00076
00077     /* Adjust range of air parcels... */
00078     if (ip0 < 0)
00079         ip0 = 0;
00080     ip0 = GSL_MIN(ip0, atm->np - 1);
00081     if (ip1 < 0)
00082         ip1 = atm->np - 1;
00083     ip1 = GSL_MIN(ip1, atm->np - 1);
00084     if (ip1 < ip0)
00085         ip1 = ip0;
00086
00087     /* Loop over air parcels... */
00088     for (ip = ip0; ip <= ip1; ip += stride) {
00089
00090         /* Check time... */
00091         if (t0 != t1)
00092             if ((t1 > t0 && (atm->time[ip] < t0 || atm->time[ip] > t1))
00093                 || (t1 < t0 && (atm->time[ip] < t0 && atm->time[ip] > t1)))
00094                 continue;
00095
00096         /* Check vertical distance... */
00097         if (p0 != p1)
00098             if ((p0 > p1 && (atm->p[ip] > p0 || atm->p[ip] < p1))
00099                 || (p0 < p1 && (atm->p[ip] > p0 && atm->p[ip] < p1)))
00100                 continue;
00101
00102         /* Check longitude... */
00103         if (lon0 != lon1)
00104             if ((lon1 > lon0 && (atm->lon[ip] < lon0 || atm->lon[ip] > lon1))
00105                 || (lon1 < lon0 && (atm->lon[ip] < lon0 && atm->lon[ip] > lon1)))
00106                 continue;
00107
00108         /* Check latitude... */
00109         if (lat0 != lat1)
00110             if ((lat1 > lat0 && (atm->lat[ip] < lat0 || atm->lat[ip] > lat1))
00111                 || (lat1 < lat0 && (atm->lat[ip] < lat0 && atm->lat[ip] > lat1)))
00112                 continue;
00113
00114         /* Check horizontal distance... */
00115         if (r0 != r1) {
00116             geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
00117             r = DIST(x0, x1);
00118             if ((r1 > r0 && (r < r0 || r > r1))
00119                 || (r1 < r0 && (r < r0 && r > r1)))
00120                 continue;
00121         }
00122
00123         /* Copy data... */
00124         atm2->time[atm2->np] = atm->time[ip];
00125         atm2->p[atm2->np] = atm->p[ip];
00126         atm2->lon[atm2->np] = atm->lon[ip];
00127         atm2->lat[atm2->np] = atm->lat[ip];
00128         for (iq = 0; iq < ctl.nq; iq++)
00129             atm2->q[iq][atm2->np] = atm->q[iq][ip];
00130         if ((++atm2->np) > NP)
00131             ERRMSG("Too many air parcels!");
00132     }
00133 }
00134
00135 /* Close file... */
00136 write_atm(argv[2], &ctl, atm2, 0);
00137
00138 /* Free... */

```

```

00139     free(atm);
00140     free(atm2);
00141
00142     return EXIT_SUCCESS;
00143 }

```

## 5.9 atm\_split.c File Reference

### Functions

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

#### 5.9.1 Detailed Description

Split air parcels into a larger number of parcels.

Definition in file [atm\\_split.c](#).

#### 5.9.2 Function Documentation

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

Definition at line 27 of file [atm\\_split.c](#).

```

00029     {
00030
00031     atm_t *atm, *atm2;
00032
00033     ctl_t ctl;
00034
00035     gsl_rng *rng;
00036
00037     FILE *in;
00038
00039     char kernel[LEN], line[LEN];
00040
00041     double dt, dx, dz, k, kk[GZ], kz[GZ], kmin, kmax, m, mmax = 0, mtot = 0,
00042            t0, t1, z, z0, z1, lon0, lon1, lat0, lat1, zmin, zmax;
00043
00044     int i, ip, iq, iz, n, nz = 0;
00045
00046     /* Allocate... */
00047     ALLOC(atm, atm_t, 1);
00048     ALLOC(atm2, atm_t, 1);
00049
00050     /* Check arguments... */
00051     if (argc < 4)
00052         ERRMSG("Give parameters: <ctl> <atm_in> <atm_out>");
00053
00054     /* Read control parameters... */
00055     read_ctl(argv[1], argc, argv, &ctl);
00056     n = (int) scan_ctl(argv[1], argc, argv, "SPLIT_N", -1, "", NULL);
00057     m = scan_ctl(argv[1], argc, argv, "SPLIT_M", -1, "-999", NULL);
00058     dt = scan_ctl(argv[1], argc, argv, "SPLIT_DT", -1, "0", NULL);
00059     t0 = scan_ctl(argv[1], argc, argv, "SPLIT_T0", -1, "0", NULL);
00060     t1 = scan_ctl(argv[1], argc, argv, "SPLIT_T1", -1, "0", NULL);
00061     dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);
00062     z0 = scan_ctl(argv[1], argc, argv, "SPLIT_Z0", -1, "0", NULL);
00063     z1 = scan_ctl(argv[1], argc, argv, "SPLIT_Z1", -1, "0", NULL);
00064     dx = scan_ctl(argv[1], argc, argv, "SPLIT_DX", -1, "0", NULL);
00065     lon0 = scan_ctl(argv[1], argc, argv, "SPLIT_LON0", -1, "0", NULL);
00066     lon1 = scan_ctl(argv[1], argc, argv, "SPLIT_LON1", -1, "0", NULL);
00067     lat0 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT0", -1, "0", NULL);

```

```

00068 lat1 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT1", -1, "0", NULL);
00069 scan_ctl(argv[1], argc, argv, "SPLIT_KERNEL", -1, "-", kernel);
00070
00071 /* Init random number generator... */
00072 gsl_rng_env_setup();
00073 rng = gsl_rng_alloc(gsl_rng_default);
00074
00075 /* Read atmospheric data... */
00076 if (!read_atm(argv[2], &ctl, atm))
00077     ERRMSG("Cannot open file!");
00078
00079 /* Read kernel function... */
00080 if (kernel[0] != '-') {
00081
00082     /* Write info... */
00083     printf("Read kernel function: %s\n", kernel);
00084
00085     /* Open file... */
00086     if (!(in = fopen(kernel, "r")))
00087         ERRMSG("Cannot open file!");
00088
00089     /* Read data... */
00090     while (fgets(line, LEN, in))
00091         if (sscanf(line, "%lg %lg", &kz[nz], &kk[nz]) == 2)
00092             if ((++nz) >= GZ)
00093                 ERRMSG("Too many height levels!");
00094
00095     /* Close file... */
00096     fclose(in);
00097
00098     /* Normalize kernel function... */
00099     zmax = gsl_stats_max(kz, 1, (size_t) nz);
00100     zmin = gsl_stats_min(kz, 1, (size_t) nz);
00101     kmax = gsl_stats_max(kk, 1, (size_t) nz);
00102     kmin = gsl_stats_min(kk, 1, (size_t) nz);
00103     for (iz = 0; iz < nz; iz++)
00104         kk[iz] = (kk[iz] - kmin) / (kmax - kmin);
00105 }
00106
00107 /* Get total and maximum mass... */
00108 if (ctl.qnt_m >= 0)
00109     for (ip = 0; ip < atm->np; ip++) {
00110         mtot += atm->q[ctl.qnt_m][ip];
00111         mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
00112     }
00113 if (m > 0)
00114     mtot = m;
00115
00116 /* Loop over air parcels... */
00117 for (i = 0; i < n; i++) {
00118
00119     /* Select air parcel... */
00120     if (ctl.qnt_m >= 0)
00121         do {
00122             ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00123             while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
00124         } else
00125             ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00126
00127     /* Set time... */
00128     if (t1 > t0)
00129         atm2->time[atm2->np] = t0 + (t1 - t0) * gsl_rng_uniform_pos(rng);
00130     else
00131         atm2->time[atm2->np] = atm->time[ip]
00132             + gsl_ran_gaussian_ziggurat(rng, dt / 2.3548);
00133
00134     /* Set vertical position... */
00135     do {
00136         if (nz > 0) {
00137             do {
00138                 z = zmin + (zmax - zmin) * gsl_rng_uniform_pos(rng);
00139                 iz = locate_irr(kz, nz, z);
00140                 k = LIN(kz[iz], kk[iz], kz[iz + 1], kk[iz + 1], z);
00141             } while (gsl_rng_uniform(rng) > k);
00142             atm2->p[atm2->np] = P(z);
00143         } else if (z1 > z0)
00144             atm2->p[atm2->np] = P(z0 + (z1 - z0) * gsl_rng_uniform_pos(rng));
00145         else
00146             atm2->p[atm2->np] = atm->p[ip]
00147                 + DZ2DP(gsl_ran_gaussian_ziggurat(rng, dz / 2.3548), atm->p[ip]);
00148     } while ((atm2->p[atm2->np] < P(100.) || atm2->p[atm2->np] > P(-1.));
00149
00150     /* Set horizontal position... */
00151     if (lon1 > lon0 && lat1 > lat0) {
00152         atm2->lon[atm2->np] = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
00153         atm2->lat[atm2->np] = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
00154     } else {

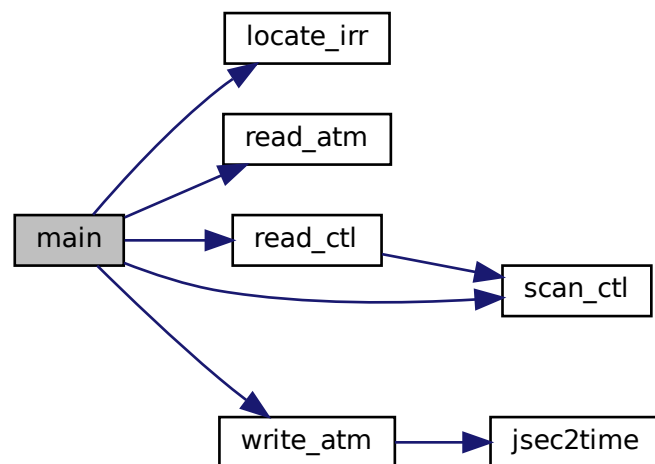
```

```

00155     atm2->lon[atm2->np] = atm->lon[ip]
00156     + gsl_ran_gaussian_ziggurat(rng, DX2DEG(dx, atm->lat[ip]) / 2.3548);
00157     atm2->lat[atm2->np] = atm->lat[ip]
00158     + gsl_ran_gaussian_ziggurat(rng, DY2DEG(dy) / 2.3548);
00159 }
00160
00161 /* Copy quantities... */
00162 for (iq = 0; iq < ctl.nq; iq++)
00163     atm2->q[iq][atm2->np] = atm->q[iq][ip];
00164
00165 /* Adjust mass... */
00166 if (ctl.qnt_m >= 0)
00167     atm2->q[ctl.qnt_m][atm2->np] = mtot / n;
00168
00169 /* Increment particle counter... */
00170 if ((++atm2->np) > NP)
00171     ERRMSG("Too many air parcels!");
00172 }
00173
00174 /* Save data and close file... */
00175 write_atm(argv[3], &ctl, atm2, 0);
00176
00177 /* Free... */
00178 free(atm);
00179 free(atm2);
00180
00181 return EXIT_SUCCESS;
00182 }

```

Here is the call graph for this function:



## 5.10 atm\_split.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
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016

```

```

00017 Copyright (C) 2013-2021 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
00037     FILE *in;
00038
00039     char kernel[LEN], line[LEN];
00040
00041     double dt, dx, dz, k, kk[GZ], kz[GZ], kmin, kmax, m, mmax = 0, mtot = 0,
00042         t0, t1, z, z0, z1, lon0, lon1, lat0, lat1, zmin, zmax;
00043
00044     int i, ip, iq, iz, n, nz = 0;
00045
00046     /* Allocate... */
00047     ALLOC(atm, atm_t, 1);
00048     ALLOC(atm2, atm_t, 1);
00049
00050     /* Check arguments... */
00051     if (argc < 4)
00052         ERRMSG("Give parameters: <ctl> <atm_in> <atm_out>");
00053
00054     /* Read control parameters... */
00055     read_ctl(argv[1], argc, argv, &ctl);
00056     n = (int) scan_ctl(argv[1], argc, argv, "SPLIT_N", -1, "", NULL);
00057     m = scan_ctl(argv[1], argc, argv, "SPLIT_M", -1, "-999", NULL);
00058     dt = scan_ctl(argv[1], argc, argv, "SPLIT_DT", -1, "0", NULL);
00059     t0 = scan_ctl(argv[1], argc, argv, "SPLIT_T0", -1, "0", NULL);
00060     t1 = scan_ctl(argv[1], argc, argv, "SPLIT_T1", -1, "0", NULL);
00061     dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);
00062     z0 = scan_ctl(argv[1], argc, argv, "SPLIT_Z0", -1, "0", NULL);
00063     z1 = scan_ctl(argv[1], argc, argv, "SPLIT_Z1", -1, "0", NULL);
00064     dx = scan_ctl(argv[1], argc, argv, "SPLIT_DX", -1, "0", NULL);
00065     lon0 = scan_ctl(argv[1], argc, argv, "SPLIT_LON0", -1, "0", NULL);
00066     lon1 = scan_ctl(argv[1], argc, argv, "SPLIT_LON1", -1, "0", NULL);
00067     lat0 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT0", -1, "0", NULL);
00068     lat1 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT1", -1, "0", NULL);
00069     scan_ctl(argv[1], argc, argv, "SPLIT_KERNEL", -1, "-", kernel);
00070
00071     /* Init random number generator... */
00072     gsl_rng_env_setup();
00073     rng = gsl_rng_alloc(gsl_rng_default);
00074
00075     /* Read atmospheric data... */
00076     if (!read_atm(argv[2], &ctl, atm))
00077         ERRMSG("Cannot open file!");
00078
00079     /* Read kernel function... */
00080     if (kernel[0] != '-') {
00081
00082         /* Write info... */
00083         printf("Read kernel function: %s\n", kernel);
00084
00085         /* Open file... */
00086         if (!(in = fopen(kernel, "r")))
00087             ERRMSG("Cannot open file!");
00088
00089         /* Read data... */
00090         while (fgets(line, LEN, in))
00091             if (sscanf(line, "%lg %lg", &kz[nz], &kk[nz]) == 2)
00092                 if ((++nz) >= GZ)
00093                     ERRMSG("Too many height levels!");
00094
00095         /* Close file... */
00096         fclose(in);
00097
00098         /* Normalize kernel function... */
00099         zmax = gsl_stats_max(kz, 1, (size_t) nz);
00100         zmin = gsl_stats_min(kz, 1, (size_t) nz);
00101         kmax = gsl_stats_max(kk, 1, (size_t) nz);
00102         kmin = gsl_stats_min(kk, 1, (size_t) nz);
00103         for (iz = 0; iz < nz; iz++)
00104             kk[iz] = (kk[iz] - kmin) / (kmax - kmin);
00105     }
00106
00107     /* Get total and maximum mass... */
00108     if (ctl.qnt_m >= 0)

```

```

00109     for (ip = 0; ip < atm->np; ip++) {
00110         mtot += atm->q[ctl.qnt_m][ip];
00111         mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
00112     }
00113     if (m > 0)
00114         mtot = m;
00115
00116     /* Loop over air parcels... */
00117     for (i = 0; i < n; i++) {
00118
00119         /* Select air parcel... */
00120         if (ctl.qnt_m >= 0)
00121             do {
00122                 ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00123                 } while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
00124             else
00125                 ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00126
00127         /* Set time... */
00128         if (t1 > t0)
00129             atm2->time[atm2->np] = t0 + (t1 - t0) * gsl_rng_uniform_pos(rng);
00130         else
00131             atm2->time[atm2->np] = atm->time[ip]
00132                 + gsl_ran_gaussian_ziggurat(rng, dt / 2.3548);
00133
00134         /* Set vertical position... */
00135         do {
00136             if (nz > 0) {
00137                 do {
00138                     z = zmin + (zmax - zmin) * gsl_rng_uniform_pos(rng);
00139                     iz = locate_irr(kz, nz, z);
00140                     k = LIN(kz[iz], kk[iz], kz[iz + 1], kk[iz + 1], z);
00141                 } while (gsl_rng_uniform(rng) > k);
00142                 atm2->p[atm2->np] = P(z);
00143             } else if (z1 > z0)
00144                 atm2->p[atm2->np] = P(z0 + (z1 - z0) * gsl_rng_uniform_pos(rng));
00145             else
00146                 atm2->p[atm2->np] = atm->p[ip]
00147                     + DZ2DP(gsl_ran_gaussian_ziggurat(rng, dz / 2.3548), atm->p[ip]);
00148             } while (atm2->p[atm2->np] < P(100.) || atm2->p[atm2->np] > P(-1.));
00149
00150         /* Set horizontal position... */
00151         if (lon1 > lon0 && lat1 > lat0) {
00152             atm2->lon[atm2->np] = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
00153             atm2->lat[atm2->np] = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
00154         } else {
00155             atm2->lon[atm2->np] = atm->lon[ip]
00156                 + gsl_ran_gaussian_ziggurat(rng, DX2DEG(dx, atm->lat[ip]) / 2.3548);
00157             atm2->lat[atm2->np] = atm->lat[ip]
00158                 + gsl_ran_gaussian_ziggurat(rng, DY2DEG(dy, atm->lat[ip]) / 2.3548);
00159         }
00160
00161         /* Copy quantities... */
00162         for (iq = 0; iq < ctl.nq; iq++)
00163             atm2->q[iq][atm2->np] = atm->q[iq][ip];
00164
00165         /* Adjust mass... */
00166         if (ctl.qnt_m >= 0)
00167             atm2->q[ctl.qnt_m][atm2->np] = mtot / n;
00168
00169         /* Increment particle counter... */
00170         if ((++atm2->np) > NP)
00171             ERRMSG("Too many air parcels!");
00172     }
00173
00174     /* Save data and close file... */
00175     write_atm(argv[3], &ctl, atm2, 0);
00176
00177     /* Free... */
00178     free(atm);
00179     free(atm2);
00180
00181     return EXIT_SUCCESS;
00182 }

```

## 5.11 atm\_stat.c File Reference

### Functions

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

### 5.11.1 Detailed Description

Calculate air parcel statistics.

Definition in file [atm\\_stat.c](#).

### 5.11.2 Function Documentation

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

Definition at line 27 of file [atm\\_stat.c](#).

```
00029     {
00030
00031     ctl_t ctl;
00032
00033     atm_t *atm, *atm_filt;
00034
00035     FILE *out;
00036
00037     char tstr[LEN];
00038
00039     double lat0, lat1, latm, lon0, lon1, lonm, p0, p1,
00040            t, t0, qm[NQ], *work, zm, *zs;
00041
00042     int ens, f, init = 0, ip, iq, year, mon, day, hour, min;
00043
00044     /* Allocate... */
00045     ALLOC(atm, atm_t, 1);
00046     ALLOC(atm_filt, atm_t, 1);
00047     ALLOC(work, double,
00048            NP);
00049     ALLOC(zs, double,
00050            NP);
00051
00052     /* Check arguments... */
00053     if (argc < 4)
00054         ERRMSG("Give parameters: <ctl> <stat.tab> <param> <atm1> [<atm2> ...]");
00055
00056     /* Read control parameters... */
00057     read_ctl(argv[1], argc, argv, &ctl);
00058     ens = (int) scan_ctl(argv[1], argc, argv, "STAT_ENS", -1, "-999", NULL);
00059     p0 = P(scan_ctl(argv[1], argc, argv, "STAT_Z0", -1, "-1000", NULL));
00060     p1 = P(scan_ctl(argv[1], argc, argv, "STAT_Z1", -1, "1000", NULL));
00061     lat0 = scan_ctl(argv[1], argc, argv, "STAT_LAT0", -1, "-1000", NULL);
00062     lat1 = scan_ctl(argv[1], argc, argv, "STAT_LAT1", -1, "1000", NULL);
00063     lon0 = scan_ctl(argv[1], argc, argv, "STAT_LON0", -1, "-1000", NULL);
00064     lon1 = scan_ctl(argv[1], argc, argv, "STAT_LON1", -1, "1000", NULL);
00065
00066     /* Write info... */
00067     printf("Write air parcel statistics: %s\n", argv[2]);
00068
00069     /* Create output file... */
00070     if (!(out = fopen(argv[2], "w")))
00071         ERRMSG("Cannot create file!");
00072
00073     /* Write header... */
00074     fprintf(out,
00075            "# $1 = time [s]\n"
00076            "# $2 = time difference [s]\n"
00077            "# $3 = altitude (%) [km]\n"
00078            "# $4 = longitude (%) [deg]\n"
00079            "# $5 = latitude (%) [deg]\n", argv[3], argv[3], argv[3]);
00080     for (iq = 0; iq < ctl.nq; iq++)
00081         fprintf(out, "# $%d = %s (%) [s]\n", iq + 6,
00082                ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq]);
00083     fprintf(out, "# $%d = number of particles\n\n", ctl.nq + 6);
00084
00085     /* Loop over files... */
00086     for (f = 4; f < argc; f++) {
00087
00088         /* Read atmospheric data... */
```

```

00089     if (!read_atm(argv[f], &ctl, atm))
00090         continue;
00091
00092     /* Get time from filename... */
00093     sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00094     year = atoi(tstr);
00095     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00096     mon = atoi(tstr);
00097     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00098     day = atoi(tstr);
00099     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00100     hour = atoi(tstr);
00101     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00102     min = atoi(tstr);
00103     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00104
00105     /* Save initial time... */
00106     if (!init) {
00107         init = 1;
00108         t0 = t;
00109     }
00110
00111     /* Filter data... */
00112     atm_filt->np = 0;
00113     for (ip = 0; ip < atm->np; ip++) {
00114
00115         /* Check time... */
00116         if (!gsl_finite(atm->time[ip]))
00117             continue;
00118
00119         /* Check ensemble index... */
00120         if (ctl.qnt_ens > 0 && atm->q[ctl.qnt_ens][ip] != ens)
00121             continue;
00122
00123         /* Check spatial range... */
00124         if (atm->p[ip] > p0 || atm->p[ip] < p1
00125             || atm->lon[ip] < lon0 || atm->lon[ip] > lon1
00126             || atm->lat[ip] < lat0 || atm->lat[ip] > lat1)
00127             continue;
00128
00129         /* Save data... */
00130         atm_filt->time[atm_filt->np] = atm->time[ip];
00131         atm_filt->p[atm_filt->np] = atm->p[ip];
00132         atm_filt->lon[atm_filt->np] = atm->lon[ip];
00133         atm_filt->lat[atm_filt->np] = atm->lat[ip];
00134         for (iq = 0; iq < ctl.nq; iq++)
00135             atm_filt->q[iq][atm_filt->np] = atm->q[iq][ip];
00136         atm_filt->np++;
00137     }
00138
00139     /* Get heights... */
00140     for (ip = 0; ip < atm_filt->np; ip++)
00141         zs[ip] = Z(atm_filt->p[ip]);
00142
00143     /* Get statistics... */
00144     if (strcasecmp(argv[3], "mean") == 0) {
00145         zm = gsl_stats_mean(zs, 1, (size_t) atm_filt->np);
00146         lonm = gsl_stats_mean(atm_filt->lon, 1, (size_t) atm_filt->np);
00147         latm = gsl_stats_mean(atm_filt->lat, 1, (size_t) atm_filt->np);
00148         for (iq = 0; iq < ctl.nq; iq++)
00149             qm[iq] = gsl_stats_mean(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00150     } else if (strcasecmp(argv[3], "stddev") == 0) {
00151         zm = gsl_stats_sd(zs, 1, (size_t) atm_filt->np);
00152         lonm = gsl_stats_sd(atm_filt->lon, 1, (size_t) atm_filt->np);
00153         latm = gsl_stats_sd(atm_filt->lat, 1, (size_t) atm_filt->np);
00154         for (iq = 0; iq < ctl.nq; iq++)
00155             qm[iq] = gsl_stats_sd(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00156     } else if (strcasecmp(argv[3], "min") == 0) {
00157         zm = gsl_stats_min(zs, 1, (size_t) atm_filt->np);
00158         lonm = gsl_stats_min(atm_filt->lon, 1, (size_t) atm_filt->np);
00159         latm = gsl_stats_min(atm_filt->lat, 1, (size_t) atm_filt->np);
00160         for (iq = 0; iq < ctl.nq; iq++)
00161             qm[iq] = gsl_stats_min(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00162     } else if (strcasecmp(argv[3], "max") == 0) {
00163         zm = gsl_stats_max(zs, 1, (size_t) atm_filt->np);
00164         lonm = gsl_stats_max(atm_filt->lon, 1, (size_t) atm_filt->np);
00165         latm = gsl_stats_max(atm_filt->lat, 1, (size_t) atm_filt->np);
00166         for (iq = 0; iq < ctl.nq; iq++)
00167             qm[iq] = gsl_stats_max(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00168     } else if (strcasecmp(argv[3], "skew") == 0) {
00169         zm = gsl_stats_skew(zs, 1, (size_t) atm_filt->np);
00170         lonm = gsl_stats_skew(atm_filt->lon, 1, (size_t) atm_filt->np);
00171         latm = gsl_stats_skew(atm_filt->lat, 1, (size_t) atm_filt->np);
00172         for (iq = 0; iq < ctl.nq; iq++)
00173             qm[iq] = gsl_stats_skew(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00174     } else if (strcasecmp(argv[3], "kurt") == 0) {
00175         zm = gsl_stats_kurtosis(zs, 1, (size_t) atm_filt->np);

```

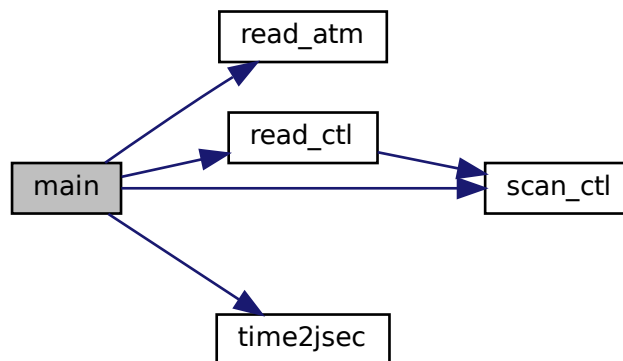


```

00176     lonm = gsl_stats_kurtosis(atm_filt->lon, 1, (size_t) atm_filt->np);
00177     latm = gsl_stats_kurtosis(atm_filt->lat, 1, (size_t) atm_filt->np);
00178     for (iq = 0; iq < ctl.nq; iq++)
00179         qm[iq] =
00180             gsl_stats_kurtosis(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00181 } else if (strcasecmp(argv[3], "median") == 0) {
00182     zm = gsl_stats_median(zs, 1, (size_t) atm_filt->np);
00183     lonm = gsl_stats_median(atm_filt->lon, 1, (size_t) atm_filt->np);
00184     latm = gsl_stats_median(atm_filt->lat, 1, (size_t) atm_filt->np);
00185     for (iq = 0; iq < ctl.nq; iq++)
00186         qm[iq] = gsl_stats_median(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00187 } else if (strcasecmp(argv[3], "absdev") == 0) {
00188     zm = gsl_stats_absdev(zs, 1, (size_t) atm_filt->np);
00189     lonm = gsl_stats_absdev(atm_filt->lon, 1, (size_t) atm_filt->np);
00190     latm = gsl_stats_absdev(atm_filt->lat, 1, (size_t) atm_filt->np);
00191     for (iq = 0; iq < ctl.nq; iq++)
00192         qm[iq] = gsl_stats_absdev(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00193 } else if (strcasecmp(argv[3], "mad") == 0) {
00194     zm = gsl_stats_mad0(zs, 1, (size_t) atm_filt->np, work);
00195     lonm = gsl_stats_mad0(atm_filt->lon, 1, (size_t) atm_filt->np, work);
00196     latm = gsl_stats_mad0(atm_filt->lat, 1, (size_t) atm_filt->np, work);
00197     for (iq = 0; iq < ctl.nq; iq++)
00198         qm[iq] =
00199             gsl_stats_mad0(atm_filt->q[iq], 1, (size_t) atm_filt->np, work);
00200 } else
00201     ERRMSG("Unknown parameter!");
00202
00203 /* Write data... */
00204 fprintf(out, "%.2f %.2f %g %g %g", t, t - t0, zm, lonm, latm);
00205 for (iq = 0; iq < ctl.nq; iq++) {
00206     fprintf(out, " ");
00207     fprintf(out, ctl.qnt_format[iq], qm[iq]);
00208 }
00209 fprintf(out, " %d\n", atm_filt->np);
00210 }
00211
00212 /* Close file... */
00213 fclose(out);
00214
00215 /* Free... */
00216 free(atm);
00217 free(atm_filt);
00218 free(work);
00219 free(zs);
00220
00221 return EXIT_SUCCESS;
00222 }

```

Here is the call graph for this function:



## 5.12 atm\_stat.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
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2019 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, *atm_filt;
00034
00035      FILE *out;
00036
00037      char tstr[LEN];
00038
00039      double lat0, lat1, latm, lon0, lon1, lonm, p0, p1,
00040             t, t0, qm[NQ], *work, zm, *zs;
00041
00042      int ens, f, init = 0, ip, iq, year, mon, day, hour, min;
00043
00044      /* Allocate... */
00045      ALLOC(atm, atm_t, 1);
00046      ALLOC(atm_filt, atm_t, 1);
00047      ALLOC(work, double,
00048             NP);
00049      ALLOC(zs, double,
00050             NP);
00051
00052      /* Check arguments... */
00053      if (argc < 4)
00054          ERRMSG("Give parameters: <ctl> <stat.tab> <param> <atm1> [<atm2> ...]");
00055
00056      /* Read control parameters... */
00057      read_ctl(argv[1], argc, argv, &ctl);
00058      ens = (int) scan_ctl(argv[1], argc, argv, "STAT_ENS", -1, "-999", NULL);
00059      p0 = P(scan_ctl(argv[1], argc, argv, "STAT_Z0", -1, "-1000", NULL));
00060      p1 = P(scan_ctl(argv[1], argc, argv, "STAT_Z1", -1, "1000", NULL));
00061      lat0 = scan_ctl(argv[1], argc, argv, "STAT_LAT0", -1, "-1000", NULL);
00062      lat1 = scan_ctl(argv[1], argc, argv, "STAT_LAT1", -1, "1000", NULL);
00063      lon0 = scan_ctl(argv[1], argc, argv, "STAT_LON0", -1, "-1000", NULL);
00064      lon1 = scan_ctl(argv[1], argc, argv, "STAT_LON1", -1, "1000", NULL);
00065
00066      /* Write info... */
00067      printf("Write air parcel statistics: %s\n", argv[2]);
00068
00069      /* Create output file... */
00070      if (!(out = fopen(argv[2], "w")))
00071          ERRMSG("Cannot create file!");
00072
00073      /* Write header... */
00074      fprintf(out,
00075             "# $1 = time [s]\n"
00076             "# $2 = time difference [s]\n"
00077             "# $3 = altitude [%s] [km]\n"
00078             "# $4 = longitude [%s] [deg]\n"
00079             "# $5 = latitude [%s] [deg]\n", argv[3], argv[3], argv[3]);
00080      for (iq = 0; iq < ctl.nq; iq++)
00081          fprintf(out, "# %d = %s [%s] [%s]\n", iq + 6,
00082                  ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq]);
00083      fprintf(out, "# %d = number of particles\n\n", ctl.nq + 6);
00084
00085      /* Loop over files... */
00086      for (f = 4; f < argc; f++) {
00087
00088          /* Read atmospheric data... */
00089          if (!read_atm(argv[f], &ctl, atm))
00090              continue;
00091
00092          /* Get time from filename... */
00093          sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);

```

```

00094     year = atoi(tstr);
00095     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00096     mon = atoi(tstr);
00097     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00098     day = atoi(tstr);
00099     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00100     hour = atoi(tstr);
00101     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00102     min = atoi(tstr);
00103     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00104
00105     /* Save intial time... */
00106     if (!init) {
00107         init = 1;
00108         t0 = t;
00109     }
00110
00111     /* Filter data... */
00112     atm_filt->np = 0;
00113     for (ip = 0; ip < atm->np; ip++) {
00114
00115         /* Check time... */
00116         if (!gsl_finite(atm->time[ip]))
00117             continue;
00118
00119         /* Check ensemble index... */
00120         if (ctl.qnt_ens > 0 && atm->q[ctl.qnt_ens][ip] != ens)
00121             continue;
00122
00123         /* Check spatial range... */
00124         if (atm->p[ip] > p0 || atm->p[ip] < p1
00125             || atm->lon[ip] < lon0 || atm->lon[ip] > lon1
00126             || atm->lat[ip] < lat0 || atm->lat[ip] > lat1)
00127             continue;
00128
00129         /* Save data... */
00130         atm_filt->time[atm_filt->np] = atm->time[ip];
00131         atm_filt->p[atm_filt->np] = atm->p[ip];
00132         atm_filt->lon[atm_filt->np] = atm->lon[ip];
00133         atm_filt->lat[atm_filt->np] = atm->lat[ip];
00134         for (iq = 0; iq < ctl.nq; iq++)
00135             atm_filt->q[iq][atm_filt->np] = atm->q[iq][ip];
00136         atm_filt->np++;
00137     }
00138
00139     /* Get heights... */
00140     for (ip = 0; ip < atm_filt->np; ip++)
00141         zs[ip] = Z(atm_filt->p[ip]);
00142
00143     /* Get statistics... */
00144     if (strcmp(argv[3], "mean") == 0) {
00145         zm = gsl_stats_mean(zs, 1, (size_t) atm_filt->np);
00146         lonm = gsl_stats_mean(atm_filt->lon, 1, (size_t) atm_filt->np);
00147         latm = gsl_stats_mean(atm_filt->lat, 1, (size_t) atm_filt->np);
00148         for (iq = 0; iq < ctl.nq; iq++)
00149             qm[iq] = gsl_stats_mean(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00150     } else if (strcmp(argv[3], "stddev") == 0) {
00151         zm = gsl_stats_sd(zs, 1, (size_t) atm_filt->np);
00152         lonm = gsl_stats_sd(atm_filt->lon, 1, (size_t) atm_filt->np);
00153         latm = gsl_stats_sd(atm_filt->lat, 1, (size_t) atm_filt->np);
00154         for (iq = 0; iq < ctl.nq; iq++)
00155             qm[iq] = gsl_stats_sd(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00156     } else if (strcmp(argv[3], "min") == 0) {
00157         zm = gsl_stats_min(zs, 1, (size_t) atm_filt->np);
00158         lonm = gsl_stats_min(atm_filt->lon, 1, (size_t) atm_filt->np);
00159         latm = gsl_stats_min(atm_filt->lat, 1, (size_t) atm_filt->np);
00160         for (iq = 0; iq < ctl.nq; iq++)
00161             qm[iq] = gsl_stats_min(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00162     } else if (strcmp(argv[3], "max") == 0) {
00163         zm = gsl_stats_max(zs, 1, (size_t) atm_filt->np);
00164         lonm = gsl_stats_max(atm_filt->lon, 1, (size_t) atm_filt->np);
00165         latm = gsl_stats_max(atm_filt->lat, 1, (size_t) atm_filt->np);
00166         for (iq = 0; iq < ctl.nq; iq++)
00167             qm[iq] = gsl_stats_max(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00168     } else if (strcmp(argv[3], "skew") == 0) {
00169         zm = gsl_stats_skew(zs, 1, (size_t) atm_filt->np);
00170         lonm = gsl_stats_skew(atm_filt->lon, 1, (size_t) atm_filt->np);
00171         latm = gsl_stats_skew(atm_filt->lat, 1, (size_t) atm_filt->np);
00172         for (iq = 0; iq < ctl.nq; iq++)
00173             qm[iq] = gsl_stats_skew(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00174     } else if (strcmp(argv[3], "kurt") == 0) {
00175         zm = gsl_stats_kurtosis(zs, 1, (size_t) atm_filt->np);
00176         lonm = gsl_stats_kurtosis(atm_filt->lon, 1, (size_t) atm_filt->np);
00177         latm = gsl_stats_kurtosis(atm_filt->lat, 1, (size_t) atm_filt->np);
00178         for (iq = 0; iq < ctl.nq; iq++)
00179             qm[iq] =
00180                 gsl_stats_kurtosis(atm_filt->q[iq], 1, (size_t) atm_filt->np);

```

```

00181     } else if (strcasecmp(argv[3], "median") == 0) {
00182         zm = gsl_stats_median(zs, 1, (size_t) atm_filt->np);
00183         lonm = gsl_stats_median(atm_filt->lon, 1, (size_t) atm_filt->np);
00184         latm = gsl_stats_median(atm_filt->lat, 1, (size_t) atm_filt->np);
00185         for (iq = 0; iq < ctl.nq; iq++)
00186             qm[iq] = gsl_stats_median(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00187     } else if (strcasecmp(argv[3], "absdev") == 0) {
00188         zm = gsl_stats_absdev(zs, 1, (size_t) atm_filt->np);
00189         lonm = gsl_stats_absdev(atm_filt->lon, 1, (size_t) atm_filt->np);
00190         latm = gsl_stats_absdev(atm_filt->lat, 1, (size_t) atm_filt->np);
00191         for (iq = 0; iq < ctl.nq; iq++)
00192             qm[iq] = gsl_stats_absdev(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00193     } else if (strcasecmp(argv[3], "mad") == 0) {
00194         zm = gsl_stats_mad0(zs, 1, (size_t) atm_filt->np, work);
00195         lonm = gsl_stats_mad0(atm_filt->lon, 1, (size_t) atm_filt->np, work);
00196         latm = gsl_stats_mad0(atm_filt->lat, 1, (size_t) atm_filt->np, work);
00197         for (iq = 0; iq < ctl.nq; iq++)
00198             qm[iq] =
00199                 gsl_stats_mad0(atm_filt->q[iq], 1, (size_t) atm_filt->np, work);
00200     } else
00201         ERRMSG("Unknown parameter!");
00202
00203     /* Write data... */
00204     fprintf(out, "%.2f %.2f %g %g %g", t, t - t0, zm, lonm, latm);
00205     for (iq = 0; iq < ctl.nq; iq++) {
00206         fprintf(out, " ");
00207         fprintf(out, ctl.qnt_format[iq], qm[iq]);
00208     }
00209     fprintf(out, " %d\n", atm_filt->np);
00210 }
00211
00212 /* Close file... */
00213 fclose(out);
00214
00215 /* Free... */
00216 free(atm);
00217 free(atm_filt);
00218 free(work);
00219 free(zs);
00220
00221 return EXIT_SUCCESS;
00222 }

```

## 5.13 day2doy.c File Reference

### Functions

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

#### 5.13.1 Detailed Description

Convert date to day of year.

Definition in file [day2doy.c](#).

#### 5.13.2 Function Documentation

```

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

```

Definition at line 27 of file [day2doy.c](#).

```

00029         {
00030
00031     int day, doy, mon, year;
00032
00033     /* Check arguments... */
00034     if (argc < 4)
00035         ERRMSG("Give parameters: <year> <mon> <day>");
00036
00037     /* Read arguments... */
00038     year = atoi(argv[1]);
00039     mon = atoi(argv[2]);
00040     day = atoi(argv[3]);
00041
00042     /* Convert... */
00043     day2doy(year, mon, day, &doy);
00044     printf("%d %d\n", year, doy);
00045
00046     return EXIT_SUCCESS;
00047 }

```

Here is the call graph for this function:



## 5.14 day2doy.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
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028     int argc,
00029     char *argv[]) {
00030
00031     int day, doy, mon, year;
00032
00033     /* Check arguments... */
00034     if (argc < 4)
00035         ERRMSG("Give parameters: <year> <mon> <day>");
00036
00037     /* Read arguments... */
00038     year = atoi(argv[1]);
00039     mon = atoi(argv[2]);
00040     day = atoi(argv[3]);
00041
00042     /* Convert... */

```

```
00043     day2doy(year, mon, day, &doy);
00044     printf("%d %d\n", year, doy);
00045
00046     return EXIT_SUCCESS;
00047 }
```

## 5.15 doy2day.c File Reference

### Functions

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

#### 5.15.1 Detailed Description

Convert day of year to date.

Definition in file [doy2day.c](#).

#### 5.15.2 Function Documentation

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

Definition at line 27 of file [doy2day.c](#).

```
00029     {
00030
00031     int day, doy, mon, year;
00032
00033     /* Check arguments... */
00034     if (argc < 3)
00035         ERRMSG("Give parameters: <year> <doy>");
00036
00037     /* Read arguments... */
00038     year = atoi(argv[1]);
00039     doy = atoi(argv[2]);
00040
00041     /* Convert... */
00042     day2doy(year, doy, &mon, &day);
00043     printf("%d %d %d\n", year, mon, day);
00044
00045     return EXIT_SUCCESS;
00046 }
```

Here is the call graph for this function:



## 5.16 doy2day.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
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013–2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028     int argc,
00029     char *argv[]) {
00030
00031     int day, doy, mon, year;
00032
00033     /* Check arguments... */
00034     if (argc < 3)
00035         ERRMSG("Give parameters: <year> <doy>");
00036
00037     /* Read arguments... */
00038     year = atoi(argv[1]);
00039     doy = atoi(argv[2]);
00040
00041     /* Convert... */
00042     doy2day(year, doy, &mon, &day);
00043     printf("%d %d %d\n", year, mon, day);
00044
00045     return EXIT_SUCCESS;
00046 }
```

## 5.17 jsec2time.c File Reference

### Functions

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

### 5.17.1 Detailed Description

Convert Julian seconds to date.

Definition in file [jsec2time.c](#).

### 5.17.2 Function Documentation

```

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

```

Definition at line 27 of file [jsec2time.c](#).

```

00029     {
00030
00031     double jsec, remain;
00032
00033     int day, hour, min, mon, sec, year;
00034
00035     /* Check arguments... */
00036     if (argc < 2)
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 %d %g\n", year, mon, day, hour, min, sec, remain);
00045
00046     return EXIT_SUCCESS;
00047 }

```

Here is the call graph for this function:



## 5.18 jsec2time.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
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2019 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 < 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 %d %g\n", year, mon, day, hour, min, sec, remain);
00045
00046     return EXIT_SUCCESS;
00047 }

```

## 5.19 lapse.c File Reference

### Functions

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

#### 5.19.1 Detailed Description

Calculate lapse rate statistics.

Definition in file [lapse.c](#).

#### 5.19.2 Function Documentation

**5.19.2.1 main()** int main (

```

    int argc,
    char * argv[] )

```

Definition at line 44 of file [lapse.c](#).

```

00046     {
00047
00048     ctl_t ctl;
00049
00050     met_t *met;
00051
00052     FILE *out;
00053
00054     static double p2[1000], t[1000], t2[1000], z[1000], z2[1000], lat_mean,
00055     z_mean;
00056
00057     static int hist_max[1000], hist_min[1000], hist_mean[1000], hist_sig[1000],
00058     nhist_max, nhist_min, nhist_mean, nhist_sig, np;
00059
00060     /* Allocate... */
00061     ALLOC(met, met_t, 1);
00062
00063     /* Check arguments... */
00064     if (argc < 4)
00065         ERRMSG("Give parameters: <ctl> <hist.tab> <met0> [ <met1> ... ]");
00066
00067     /* Read control parameters... */
00068     read_ctl(argv[1], argc, argv, &ctl);
00069     int dz = (int) scan_ctl(argv[1], argc, argv, "LAPSE_DZ", -1, "20", NULL);
00070     double lat0 =
00071         (int) scan_ctl(argv[1], argc, argv, "LAPSE_LAT0", -1, "-90", NULL);
00072     double lat1 =
00073         (int) scan_ctl(argv[1], argc, argv, "LAPSE_LAT1", -1, "90", NULL);
00074     double z0 = (int) scan_ctl(argv[1], argc, argv, "LAPSE_Z0", -1, "0", NULL);
00075     double z1 =
00076         (int) scan_ctl(argv[1], argc, argv, "LAPSE_Z1", -1, "100", NULL);
00077
00078     /* Loop over files... */
00079     for (int i = 3; i < argc; i++) {
00080
00081         /* Read meteorological data... */
00082         if (!read_met(&ctl, argv[i], met))
00083             continue;
00084

```

```

00085     /* Get altitude and pressure profiles... */
00086     for (int iz = 0; iz < met->np; iz++)
00087         z[iz] = Z(met->p[iz]);
00088     for (int iz = 0; iz <= 250; iz++) {
00089         z2[iz] = 0.0 + 0.1 * iz;
00090         p2[iz] = P(z2[iz]);
00091     }
00092
00093     /* Loop over grid points... */
00094     for (int ix = 0; ix < met->nx; ix++)
00095         for (int iy = 0; iy < met->ny; iy++) {
00096
00097             /* Check latitude range... */
00098             if (met->lat[iy] < lat0 || met->lat[iy] > lat1)
00099                 continue;
00100
00101             /* Interpolate temperature profile... */
00102             for (int iz = 0; iz < met->np; iz++)
00103                 t[iz] = met->t[ix][iy][iz];
00104             spline(z, t, met->np, z2, t2, 251);
00105
00106             /* Loop over vertical levels... */
00107             for (int iz = 0; iz <= 250; iz++) {
00108
00109                 /* Check height range... */
00110                 if (z2[iz] < z0 || z2[iz] > z1)
00111                     continue;
00112
00113                 /* Check surface pressure... */
00114                 if (p2[iz] > met->ps[ix][iy])
00115                     continue;
00116
00117                 /* Get mean latitude and height... */
00118                 lat_mean += met->lat[iy];
00119                 z_mean += z2[iz];
00120                 np++;
00121
00122                 /* Get lapse rates within a vertical layer... */
00123                 int nlapse = 0;
00124                 double lapse_max = -1e99, lapse_min = 1e99, lapse_mean =
00125                     0, lapse_sig = 0;
00126                 for (int iz2 = iz + 1; iz2 <= iz + dz; iz2++) {
00127                     lapse_max =
00128                         GSL_MAX(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_max);
00129                     lapse_min =
00130                         GSL_MIN(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_min);
00131                     lapse_mean += LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]);
00132                     lapse_sig += SQR(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]));
00133                     nlapse++;
00134                 }
00135                 lapse_mean /= nlapse;
00136                 lapse_sig = sqrt(GSL_MAX(lapse_sig / nlapse - SQR(lapse_mean), 0));
00137
00138                 /* Get histograms... */
00139                 int idx = (int) ((lapse_max - LAPSEMIN) / DLAPSE);
00140                 if (idx >= 0 && idx < IDXMAX) {
00141                     hist_max[idx]++;
00142                     nhist_max++;
00143                 }
00144
00145                 idx = (int) ((lapse_min - LAPSEMIN) / DLAPSE);
00146                 if (idx >= 0 && idx < IDXMAX) {
00147                     hist_min[idx]++;
00148                     nhist_min++;
00149                 }
00150
00151                 idx = (int) ((lapse_mean - LAPSEMIN) / DLAPSE);
00152                 if (idx >= 0 && idx < IDXMAX) {
00153                     hist_mean[idx]++;
00154                     nhist_mean++;
00155                 }
00156
00157                 idx = (int) ((lapse_sig - LAPSEMIN) / DLAPSE);
00158                 if (idx >= 0 && idx < IDXMAX) {
00159                     hist_sig[idx]++;
00160                     nhist_sig++;
00161                 }
00162             }
00163         }
00164     }
00165
00166     /* Create output file... */
00167     printf("Write lapse rate data: %s\n", argv[2]);
00168     if (!(out = fopen(argv[2], "w")))
00169         ERRMSG("Cannot create file!");
00170
00171     /* Write header... */

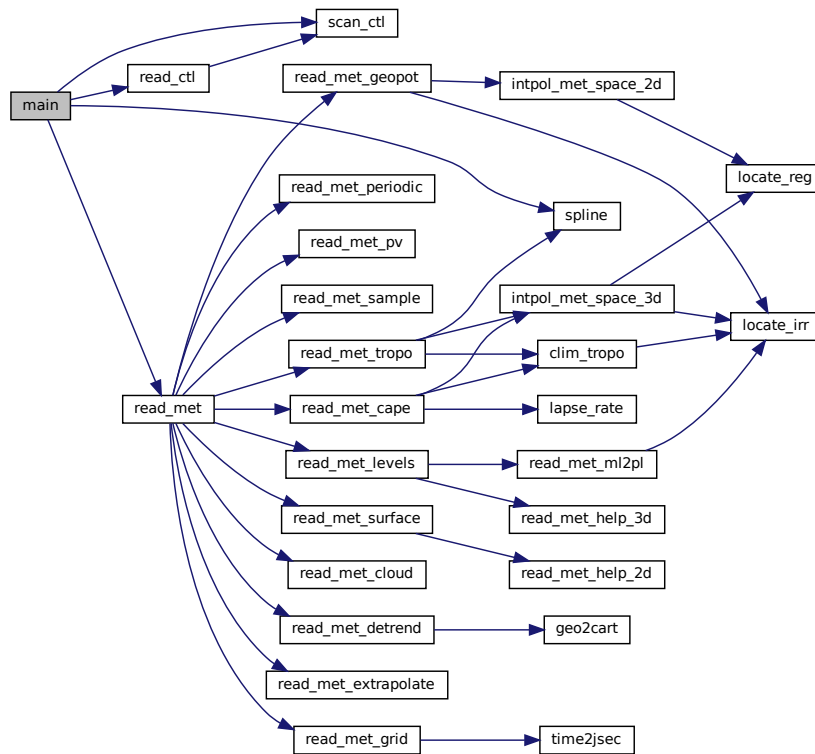
```

```

00172 fprintf(out,
00173     "# $1 = mean altitude [km]\n"
00174     "# $2 = mean latitude [deg]\n"
00175     "# $3 = lapse rate [K/km]\n"
00176     "# $4 = counts of maxima per bin\n"
00177     "# $5 = total number of maxima\n"
00178     "# $6 = normalized frequency of maxima\n"
00179     "# $7 = counts of minima per bin\n"
00180     "# $8 = total number of minima\n"
00181     "# $9 = normalized frequency of minima\n"
00182     "# $10 = counts of means per bin\n"
00183     "# $11 = total number of means\n"
00184     "# $12 = normalized frequency of means\n"
00185     "# $13 = counts of sigmas per bin\n"
00186     "# $14 = total number of sigmas\n"
00187     "# $15 = normalized frequency of sigmas\n\n");
00188
00189 /* Write data... */
00190 double nmax_max = 0, nmax_min = 0, nmax_mean = 0, nmax_sig = 0;
00191 for (int idx = 0; idx < IDXMAX; idx++) {
00192     nmax_max = GSL_MAX(hist_max[idx], nmax_max);
00193     nmax_min = GSL_MAX(hist_min[idx], nmax_min);
00194     nmax_mean = GSL_MAX(hist_mean[idx], nmax_mean);
00195     nmax_sig = GSL_MAX(hist_sig[idx], nmax_sig);
00196 }
00197
00198 for (int idx = 0; idx < IDXMAX; idx++)
00199     fprintf(out,
00200         "%g %g %g %d %d %g %d %d %g %d %d %g\n",
00201         z_mean / np, lat_mean / np, (idx + .5) * DLAPSE + LAPSEMIN,
00202         hist_max[idx], nhist_max,
00203         (double) hist_max[idx] / (double) nmax_max, hist_min[idx],
00204         nhist_min, (double) hist_min[idx] / (double) nmax_min,
00205         hist_mean[idx], nhist_mean,
00206         (double) hist_mean[idx] / (double) nmax_mean, hist_sig[idx],
00207         nhist_sig, (double) hist_sig[idx] / (double) nmax_sig);
00208
00209 /* Close file... */
00210 fclose(out);
00211
00212 /* Free... */
00213 free(met);
00214
00215 return EXIT_SUCCESS;
00216 }

```

Here is the call graph for this function:



## 5.20 lapse.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
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028  Dimensions...
00029  ----- */
00030
00032 #define LAPSEMIN -20.0
00033
00035 #define DLAPSE 0.1
00036
00038 #define IDXMAX 400
00039
00040 /* -----
00041  Main...
00042  ----- */
00043
00044 int main(
00045     int argc,

```

```

00046 char *argv[] {
00047
00048     ctl_t ctl;
00049
00050     met_t *met;
00051
00052     FILE *out;
00053
00054     static double p2[1000], t[1000], t2[1000], z[1000], z2[1000], lat_mean,
00055         z_mean;
00056
00057     static int hist_max[1000], hist_min[1000], hist_mean[1000], hist_sig[1000],
00058         nhist_max, nhist_min, nhist_mean, nhist_sig, np;
00059
00060     /* Allocate... */
00061     ALLOC(met, met_t, 1);
00062
00063     /* Check arguments... */
00064     if (argc < 4)
00065         ERRMSG("Give parameters: <ctl> <hist.tab> <met0> [ <met1> ... ]");
00066
00067     /* Read control parameters... */
00068     read_ctl(argv[1], argc, argv, &ctl);
00069     int dz = (int) scan_ctl(argv[1], argc, argv, "LAPSE_DZ", -1, "20", NULL);
00070     double lat0 =
00071         (int) scan_ctl(argv[1], argc, argv, "LAPSE_LAT0", -1, "-90", NULL);
00072     double lat1 =
00073         (int) scan_ctl(argv[1], argc, argv, "LAPSE_LAT1", -1, "90", NULL);
00074     double z0 = (int) scan_ctl(argv[1], argc, argv, "LAPSE_Z0", -1, "0", NULL);
00075     double z1 =
00076         (int) scan_ctl(argv[1], argc, argv, "LAPSE_Z1", -1, "100", NULL);
00077
00078     /* Loop over files... */
00079     for (int i = 3; i < argc; i++) {
00080
00081         /* Read meteorological data... */
00082         if (!read_met(&ctl, argv[i], met))
00083             continue;
00084
00085         /* Get altitude and pressure profiles... */
00086         for (int iz = 0; iz < met->np; iz++)
00087             z[iz] = Z(met->p[iz]);
00088         for (int iz = 0; iz <= 250; iz++) {
00089             z2[iz] = 0.0 + 0.1 * iz;
00090             p2[iz] = P(z2[iz]);
00091         }
00092
00093         /* Loop over grid points... */
00094         for (int ix = 0; ix < met->nx; ix++)
00095             for (int iy = 0; iy < met->ny; iy++) {
00096
00097                 /* Check latitude range... */
00098                 if (met->lat[iy] < lat0 || met->lat[iy] > lat1)
00099                     continue;
00100
00101                 /* Interpolate temperature profile... */
00102                 for (int iz = 0; iz < met->np; iz++)
00103                     t[iz] = met->t[ix][iy][iz];
00104                 spline(z, t, met->np, z2, t2, 251);
00105
00106                 /* Loop over vertical levels... */
00107                 for (int iz = 0; iz <= 250; iz++) {
00108
00109                     /* Check height range... */
00110                     if (z2[iz] < z0 || z2[iz] > z1)
00111                         continue;
00112
00113                     /* Check surface pressure... */
00114                     if (p2[iz] > met->ps[ix][iy])
00115                         continue;
00116
00117                     /* Get mean latitude and height... */
00118                     lat_mean += met->lat[iy];
00119                     z_mean += z2[iz];
00120                     np++;
00121
00122                     /* Get lapse rates within a vertical layer... */
00123                     int nlapse = 0;
00124                     double lapse_max = -1e99, lapse_min = 1e99, lapse_mean =
00125                         0, lapse_sig = 0;
00126                     for (int iz2 = iz + 1; iz2 <= iz + dz; iz2++) {
00127                         lapse_max =
00128                             GSL_MAX(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_max);
00129                         lapse_min =
00130                             GSL_MIN(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_min);
00131                         lapse_mean += LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]);
00132                         lapse_sig += SQR(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]));

```

```

00133         nlapse++;
00134     }
00135     lapse_mean /= nlapse;
00136     lapse_sig = sqrt(GSL_MAX(lapse_sig / nlapse - SQR(lapse_mean), 0));
00137
00138     /* Get histograms... */
00139     int idx = (int) ((lapse_max - LAPSEMIN) / DLAPSE);
00140     if (idx >= 0 && idx < IDXMAX) {
00141         hist_max[idx]++;
00142         nhist_max++;
00143     }
00144
00145     idx = (int) ((lapse_min - LAPSEMIN) / DLAPSE);
00146     if (idx >= 0 && idx < IDXMAX) {
00147         hist_min[idx]++;
00148         nhist_min++;
00149     }
00150
00151     idx = (int) ((lapse_mean - LAPSEMIN) / DLAPSE);
00152     if (idx >= 0 && idx < IDXMAX) {
00153         hist_mean[idx]++;
00154         nhist_mean++;
00155     }
00156
00157     idx = (int) ((lapse_sig - LAPSEMIN) / DLAPSE);
00158     if (idx >= 0 && idx < IDXMAX) {
00159         hist_sig[idx]++;
00160         nhist_sig++;
00161     }
00162 }
00163 }
00164 }
00165
00166 /* Create output file... */
00167 printf("Write lapse rate data: %s\n", argv[2]);
00168 if (!(out = fopen(argv[2], "w")))
00169     ERRMSG("Cannot create file!");
00170
00171 /* Write header... */
00172 fprintf(out,
00173     "# $1 = mean altitude [km]\n"
00174     "# $2 = mean latitude [deg]\n"
00175     "# $3 = lapse rate [K/km]\n"
00176     "# $4 = counts of maxima per bin\n"
00177     "# $5 = total number of maxima\n"
00178     "# $6 = normalized frequency of maxima\n"
00179     "# $7 = counts of minima per bin\n"
00180     "# $8 = total number of minima\n"
00181     "# $9 = normalized frequency of minima\n"
00182     "# $10 = counts of means per bin\n"
00183     "# $11 = total number of means\n"
00184     "# $12 = normalized frequency of means\n"
00185     "# $13 = counts of sigmas per bin\n"
00186     "# $14 = total number of sigmas\n"
00187     "# $15 = normalized frequency of sigmas\n\n");
00188
00189 /* Write data... */
00190 double nmax_max = 0, nmax_min = 0, nmax_mean = 0, nmax_sig = 0;
00191 for (int idx = 0; idx < IDXMAX; idx++) {
00192     nmax_max = GSL_MAX(hist_max[idx], nmax_max);
00193     nmax_min = GSL_MAX(hist_min[idx], nmax_min);
00194     nmax_mean = GSL_MAX(hist_mean[idx], nmax_mean);
00195     nmax_sig = GSL_MAX(hist_sig[idx], nmax_sig);
00196 }
00197
00198 for (int idx = 0; idx < IDXMAX; idx++)
00199     fprintf(out,
00200         "%g %g %g %d %d %g %d %d %g %d %d %g %d %d %g\n",
00201         z_mean / np, lat_mean / np, (idx + .5) * DLAPSE + LAPSEMIN,
00202         hist_max[idx], nhist_max,
00203         (double) hist_max[idx] / (double) nmax_max, hist_min[idx],
00204         nhist_min, (double) hist_min[idx] / (double) nmax_min,
00205         hist_mean[idx], nhist_mean,
00206         (double) hist_mean[idx] / (double) nmax_mean, hist_sig[idx],
00207         nhist_sig, (double) hist_sig[idx] / (double) nmax_sig);
00208
00209 /* Close file... */
00210 fclose(out);
00211
00212 /* Free... */
00213 free(met);
00214
00215 return EXIT_SUCCESS;
00216 }

```

## 5.21 libtrac.c File Reference

### Functions

- void [cart2geo](#) (double \*x, double \*z, double \*lon, double \*lat)  
*Convert Cartesian coordinates to geolocation.*
- double [clim\\_hno3](#) (double t, double lat, double p)  
*Climatology of HNO<sub>3</sub> volume mixing ratios.*
- double [clim\\_oh](#) (double t, double lat, double p)  
*Climatology of OH number concentrations.*
- double [clim\\_tropo](#) (double t, double lat)  
*Climatology of tropopause pressure.*
- void [day2doy](#) (int year, int mon, int day, int \*doy)  
*Get day of year from date.*
- void [doy2day](#) (int year, int doy, int \*mon, int \*day)  
*Get date from day of year.*
- void [geo2cart](#) (double z, double lon, double lat, double \*x)  
*Convert geolocation to Cartesian coordinates.*
- void [get\\_met](#) (ctl\_t \*ctl, double t, met\_t \*\*met0, met\_t \*\*met1)  
*Get meteorological data for given time step.*
- void [get\\_met\\_help](#) (double t, int direct, char \*metbase, double dt\_met, char \*filename)  
*Get meteorological data for time step.*
- void [get\\_met\\_replace](#) (char \*orig, char \*search, char \*repl)  
*Replace template strings in filename.*
- void [intpol\\_met\\_space\\_3d](#) (met\_t \*met, float array[EX][EY][EP], double p, double lon, double lat, double \*var, int \*ci, double \*cw, int init)  
*Spatial interpolation of meteorological data.*
- void [intpol\\_met\\_space\\_2d](#) (met\_t \*met, float array[EX][EY], double lon, double lat, double \*var, int \*ci, double \*cw, int init)  
*Spatial interpolation of meteorological data.*
- void [intpol\\_met\\_time\\_3d](#) (met\_t \*met0, float array0[EX][EY][EP], met\_t \*met1, float array1[EX][EY][EP], double ts, double p, double lon, double lat, double \*var, int \*ci, double \*cw, int init)  
*Temporal interpolation of meteorological data.*
- void [intpol\\_met\\_time\\_2d](#) (met\_t \*met0, float array0[EX][EY], met\_t \*met1, float array1[EX][EY], double ts, double lon, double lat, double \*var, int \*ci, double \*cw, int init)  
*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.*
- double [lapse\\_rate](#) (double t, double h2o)  
*Calculate moist adiabatic lapse rate.*
- int [locate\\_irr](#) (double \*xx, int n, double x)  
*Find array index for irregular grid.*
- int [locate\\_reg](#) (double \*xx, int n, double x)  
*Find array index for regular grid.*
- double [nat\\_temperature](#) (double p, double h2o, double hno3)  
*Calculate NAT existence temperature.*
- int [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.*
- int [read\\_met](#) (ctl\_t \*ctl, char \*filename, met\_t \*met)

- Read meteorological data file.*
- void [read\\_met\\_cape](#) ([met\\_t](#) \*met)
- Calculate convective available potential energy.*
- void [read\\_met\\_cloud](#) ([met\\_t](#) \*met)
- Calculate cloud properties.*
- void [read\\_met\\_detrend](#) ([ctl\\_t](#) \*ctl, [met\\_t](#) \*met)
- Apply detrending method to temperature and winds.*
- void [read\\_met\\_extrapolate](#) ([met\\_t](#) \*met)
- Extrapolate meteorological data at lower boundary.*
- void [read\\_met\\_geopot](#) ([ctl\\_t](#) \*ctl, [met\\_t](#) \*met)
- Calculate geopotential heights.*
- void [read\\_met\\_grid](#) (char \*filename, int ncid, [ctl\\_t](#) \*ctl, [met\\_t](#) \*met)
- Read coordinates of meteorological data.*
- int [read\\_met\\_help\\_3d](#) (int ncid, char \*varname, char \*varname2, [met\\_t](#) \*met, float dest[EX][EY][EP], float scl)
- Read and convert 3D variable from meteorological data file.*
- int [read\\_met\\_help\\_2d](#) (int ncid, char \*varname, char \*varname2, [met\\_t](#) \*met, float dest[EX][EY], float scl)
- Read and convert 2D variable from meteorological data file.*
- void [read\\_met\\_levels](#) (int ncid, [ctl\\_t](#) \*ctl, [met\\_t](#) \*met)
- 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.*
- void [read\\_met\\_pv](#) ([met\\_t](#) \*met)
- Calculate potential vorticity.*
- void [read\\_met\\_sample](#) ([ctl\\_t](#) \*ctl, [met\\_t](#) \*met)
- Downsampling of meteorological data.*
- void [read\\_met\\_surface](#) (int ncid, [met\\_t](#) \*met)
- Read surface data.*
- void [read\\_met\\_tropo](#) ([ctl\\_t](#) \*ctl, [met\\_t](#) \*met)
- Calculate tropopause data.*
- 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.*
- double [sedi](#) (double p, double T, double r\_p, double rho\_p)
- Calculate sedimentation velocity.*
- void [spline](#) (double \*x, double \*y, int n, double \*x2, double \*y2, int n2)
- Spline interpolation.*
- double [stddev](#) (double \*data, int n)
- Calculate standard deviation.*
- 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 output)
- Measure wall-clock time.*
- 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\\_sample](#) (const char \*filename, [ctl\\_t](#) \*ctl, [met\\_t](#) \*met0, [met\\_t](#) \*met1, [atm\\_t](#) \*atm, double t)

*Write sample data.*

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

*Write station data.*

### 5.21.1 Detailed Description

MPTRAC library definitions.

Definition in file [libtrac.c](#).

### 5.21.2 Function Documentation

#### 5.21.2.1 [cart2geo\(\)](#)

```
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 = NORM(x);
00036     *lat = asin(x[2] / radius) * 180. / M_PI;
00037     *lon = atan2(x[1], x[0]) * 180. / M_PI;
00038     *z = radius - RE;
00039 }
```

#### 5.21.2.2 [clim\\_hno3\(\)](#)

```
double clim_hno3 (
    double t,
    double lat,
    double p )
```

Climatology of HNO3 volume mixing ratios.

Definition at line 295 of file [libtrac.c](#).

```
00298     {
00299
00300     /* Get seconds since begin of year... */
00301     double sec = FMOD(t, 365.25 * 86400.);
00302     while (sec < 0)
00303         sec += 365.25 * 86400.;
00304
00305     /* Check pressure... */
00306     if (p < clim_hno3_ps[0])
00307         p = clim_hno3_ps[0];
00308     else if (p > clim_hno3_ps[9])
00309         p = clim_hno3_ps[9];
00310
00311     /* Check latitude... */
```

```

00312     if (lat < clim_hno3_lats[0])
00313         lat = clim_hno3_lats[0];
00314     else if (lat > clim_hno3_lats[17])
00315         lat = clim_hno3_lats[17];
00316
00317     /* Get indices... */
00318     int isec = locate_irr(clim_hno3_secs, 12, sec);
00319     int ilat = locate_reg(clim_hno3_lats, 18, lat);
00320     int ip = locate_irr(clim_hno3_ps, 10, p);
00321
00322     /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
00323     double aux00 = LIN(clim_hno3_ps[ip],
00324                        clim_hno3_var[isec][ilat][ip],
00325                        clim_hno3_ps[ip + 1],
00326                        clim_hno3_var[isec][ilat][ip + 1], p);
00327     double aux01 = LIN(clim_hno3_ps[ip],
00328                        clim_hno3_var[isec][ilat + 1][ip],
00329                        clim_hno3_ps[ip + 1],
00330                        clim_hno3_var[isec][ilat + 1][ip + 1], p);
00331     double aux10 = LIN(clim_hno3_ps[ip],
00332                        clim_hno3_var[isec + 1][ilat][ip],
00333                        clim_hno3_ps[ip + 1],
00334                        clim_hno3_var[isec + 1][ilat][ip + 1], p);
00335     double aux11 = LIN(clim_hno3_ps[ip],
00336                        clim_hno3_var[isec + 1][ilat + 1][ip],
00337                        clim_hno3_ps[ip + 1],
00338                        clim_hno3_var[isec + 1][ilat + 1][ip + 1], p);
00339     aux00 = LIN(clim_hno3_lats[ilat], aux00,
00340                clim_hno3_lats[ilat + 1], aux01, lat);
00341     aux11 = LIN(clim_hno3_lats[ilat], aux10,
00342                clim_hno3_lats[ilat + 1], aux11, lat);
00343     aux00 = LIN(clim_hno3_secs[isec], aux00,
00344                clim_hno3_secs[isec + 1], aux11, sec);
00345     return GSL_MAX(aux00, 0.0);
00346 }

```

**5.21.2.3 clim\_oh()** double clim\_oh (  
double t,  
double lat,  
double p )

Climatology of OH number concentrations.

Definition at line 1329 of file libtrac.c.

```

01332     {
01333
01334     /* Get seconds since begin of year... */
01335     double sec = FMOD(t, 365.25 * 86400.);
01336     while (sec < 0)
01337         sec += 365.25 * 86400.;
01338
01339     /* Check pressure... */
01340     if (p < clim_oh_ps[0])
01341         p = clim_oh_ps[0];
01342     else if (p > clim_oh_ps[33])
01343         p = clim_oh_ps[33];
01344
01345     /* Check latitude... */
01346     if (lat < clim_oh_lats[0])
01347         lat = clim_oh_lats[0];
01348     else if (lat > clim_oh_lats[17])
01349         lat = clim_oh_lats[17];
01350
01351     /* Get indices... */
01352     int isec = locate_irr(clim_oh_secs, 12, sec);
01353     int ilat = locate_reg(clim_oh_lats, 18, lat);
01354     int ip = locate_irr(clim_oh_ps, 34, p);
01355
01356     /* Interpolate OH climatology (Pommrich et al., 2014)... */
01357     double aux00 = LIN(clim_oh_ps[ip],
01358                        clim_oh_var[isec][ilat][ip],
01359                        clim_oh_ps[ip + 1],
01360                        clim_oh_var[isec][ilat][ip + 1], p);
01361     double aux01 = LIN(clim_oh_ps[ip],
01362                        clim_oh_var[isec][ilat + 1][ip],
01363                        clim_oh_ps[ip + 1],
01364                        clim_oh_var[isec][ilat + 1][ip + 1], p);
01365     double aux10 = LIN(clim_oh_ps[ip],

```

```

01366             clim_oh_var[isec + 1][ilat][ip],
01367             clim_oh_ps[ip + 1],
01368             clim_oh_var[isec + 1][ilat][ip + 1], p);
01369     double aux11 = LIN(clim_oh_ps[ip],
01370             clim_oh_var[isec + 1][ilat + 1][ip],
01371             clim_oh_ps[ip + 1],
01372             clim_oh_var[isec + 1][ilat + 1][ip + 1], p);
01373     aux00 = LIN(clim_oh_lats[ilat], aux00, clim_oh_lats[ilat + 1], aux01, lat);
01374     aux11 = LIN(clim_oh_lats[ilat], aux10, clim_oh_lats[ilat + 1], aux11, lat);
01375     aux00 = LIN(clim_oh_secs[isec], aux00, clim_oh_secs[isec + 1], aux11, sec);
01376     return GSL_MAX(1e6 * aux00, 0.0);
01377 }

```

**5.21.2.4 clim\_tropo()** double clim\_tropo (  
double t,  
double lat )

Climatology of tropopause pressure.

Definition at line 1510 of file libtrac.c.

```

01512     {
01513
01514     /* Get seconds since begin of year... */
01515     double sec = FMOD(t, 365.25 * 86400.);
01516     while (sec < 0)
01517         sec += 365.25 * 86400.;
01518
01519     /* Get indices... */
01520     int isec = locate_irr(clim_tropo_secs, 12, sec);
01521     int ilat = locate_reg(clim_tropo_lats, 73, lat);
01522
01523     /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
01524     double p0 = LIN(clim_tropo_lats[ilat],
01525             clim_tropo_tps[isec][ilat],
01526             clim_tropo_lats[ilat + 1],
01527             clim_tropo_tps[isec][ilat + 1], lat);
01528     double p1 = LIN(clim_tropo_lats[ilat],
01529             clim_tropo_tps[isec + 1][ilat],
01530             clim_tropo_lats[ilat + 1],
01531             clim_tropo_tps[isec + 1][ilat + 1], lat);
01532     return LIN(clim_tropo_secs[isec], p0, clim_tropo_secs[isec + 1], p1, sec);
01533 }

```

Here is the call graph for this function:



**5.21.2.5 day2doy()** void day2doy (  
int year,  
int mon,  
int day,  
int \* doy )

Get day of year from date.

Definition at line 1537 of file [libtrac.c](#).

```
01541     {
01542
01543     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01544     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01545
01546     /* Get day of year... */
01547     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
01548         *doy = d0l[mon - 1] + day - 1;
01549     else
01550         *doy = d0[mon - 1] + day - 1;
01551 }
```

#### 5.21.2.6 doy2day() void doy2day (

```
    int year,
    int doy,
    int * mon,
    int * day )
```

Get date from day of year.

Definition at line 1555 of file [libtrac.c](#).

```
01559     {
01560
01561     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01562     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01563     int i;
01564
01565     /* Get month and day... */
01566     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
01567         for (i = 11; i >= 0; i--)
01568             if (d0l[i] <= doy)
01569                 break;
01570         *mon = i + 1;
01571         *day = doy - d0l[i] + 1;
01572     } else {
01573         for (i = 11; i >= 0; i--)
01574             if (d0[i] <= doy)
01575                 break;
01576         *mon = i + 1;
01577         *day = doy - d0[i] + 1;
01578     }
01579 }
```

#### 5.21.2.7 geo2cart() void geo2cart (

```
    double z,
    double lon,
    double lat,
    double * x )
```

Convert geolocation to Cartesian coordinates.

Definition at line 1583 of file [libtrac.c](#).

```
01587     {
01588
01589     double radius = z + RE;
01590     x[0] = radius * cos(lat / 180. * M_PI) * cos(lon / 180. * M_PI);
01591     x[1] = radius * cos(lat / 180. * M_PI) * sin(lon / 180. * M_PI);
01592     x[2] = radius * sin(lat / 180. * M_PI);
01593 }
```

```

5.21.2.8 get_met() void get_met (
    ctl_t * ctl,
    double t,
    met_t ** met0,
    met_t ** met1 )

```

Get meteorological data for given time step.

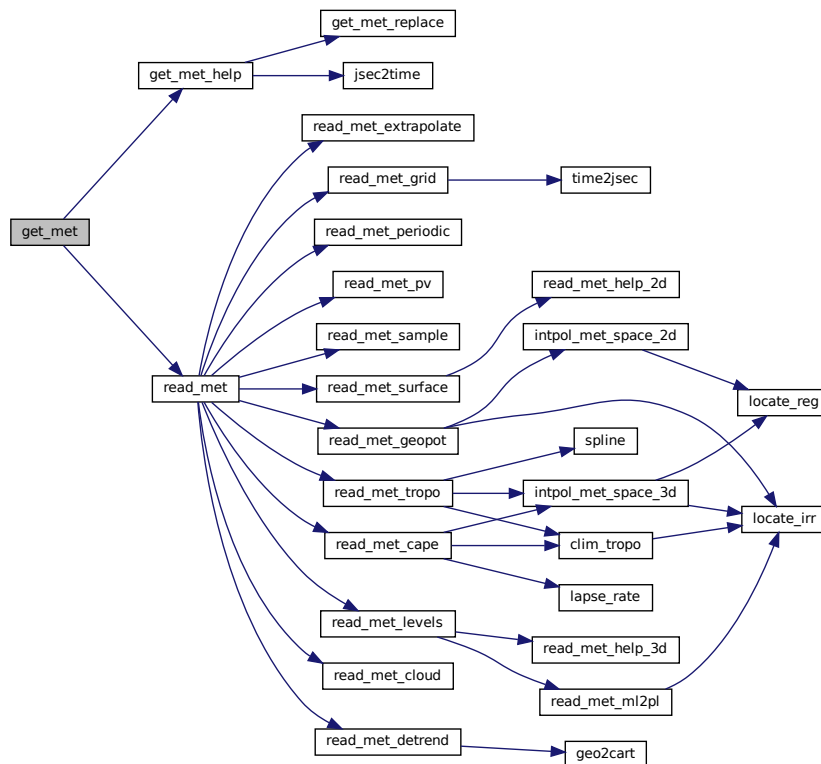
Definition at line 1597 of file libtrac.c.

```

01601     {
01602
01603     static int init, ip, ix, iy;
01604
01605     met_t *mets;
01606
01607     char filename[LEN];
01608
01609     /* Set timer... */
01610     SELECT_TIMER("GET_MET", NVTX_READ);
01611
01612     /* Init... */
01613     if (t == ctl->t_start || !init) {
01614         init = 1;
01615
01616         get_met_help(t, -1, ctl->metbase, ctl->dt_met, filename);
01617         if (!read_met(ctl, filename, *met0))
01618             ERRMSG("Cannot open file!");
01619
01620         get_met_help(t + 1.0 * ctl->direction, 1, ctl->metbase, ctl->dt_met,
01621                     filename);
01622         if (!read_met(ctl, filename, *met1))
01623             ERRMSG("Cannot open file!");
01624 #ifdef _OPENACC
01625         met_t *met0up = *met0;
01626         met_t *met1up = *met1;
01627 #pragma acc update device(met0up[:1],met1up[:1])
01628 #endif
01629     }
01630
01631     /* Read new data for forward trajectories... */
01632     if (t > (*met1)->time && ctl->direction == 1) {
01633         mets = *met1;
01634         *met1 = *met0;
01635         *met0 = mets;
01636         get_met_help(t, 1, ctl->metbase, ctl->dt_met, filename);
01637         if (!read_met(ctl, filename, *met1))
01638             ERRMSG("Cannot open file!");
01639 #ifdef _OPENACC
01640         met_t *met1up = *met1;
01641 #pragma acc update device(met1up[:1])
01642 #endif
01643     }
01644
01645     /* Read new data for backward trajectories... */
01646     if (t < (*met0)->time && ctl->direction == -1) {
01647         mets = *met1;
01648         *met1 = *met0;
01649         *met0 = mets;
01650         get_met_help(t, -1, ctl->metbase, ctl->dt_met, filename);
01651         if (!read_met(ctl, filename, *met0))
01652             ERRMSG("Cannot open file!");
01653 #ifdef _OPENACC
01654         met_t *met0up = *met0;
01655 #pragma acc update device(met0up[:1])
01656 #endif
01657     }
01658
01659     /* Check that grids are consistent... */
01660     if (((*met0)->nx != (*met1)->nx
01661         || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
01662         ERRMSG("Meteo grid dimensions do not match!");
01663     for (ix = 0; ix < (*met0)->nx; ix++)
01664         if (fabs((*met0)->lon[ix] - (*met1)->lon[ix]) > 0.001)
01665             ERRMSG("Meteo grid longitudes do not match!");
01666     for (iy = 0; iy < (*met0)->ny; iy++)
01667         if (fabs((*met0)->lat[iy] - (*met1)->lat[iy]) > 0.001)
01668             ERRMSG("Meteo grid latitudes do not match!");
01669     for (ip = 0; ip < (*met0)->np; ip++)
01670         if (fabs((*met0)->p[ip] - (*met1)->p[ip]) > 0.001)
01671             ERRMSG("Meteo grid pressure levels do not match!");
01672 }

```

Here is the call graph for this function:



**5.21.2.9 get\_met\_help()** void get\_met\_help (

```

double t,
int direct,
char * metbase,
double dt_met,
char * filename )

```

Get meteorological data for time step.

Definition at line 1676 of file libtrac.c.

```

01681     {
01682
01683     char repl[LEN];
01684
01685     double t6, r;
01686
01687     int year, mon, day, hour, min, sec;
01688
01689     /* Round time to fixed intervals... */
01690     if (direct == -1)
01691         t6 = floor(t / dt_met) * dt_met;
01692     else
01693         t6 = ceil(t / dt_met) * dt_met;
01694
01695     /* Decode time... */
01696     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
01697
01698     /* Set filename... */
01699     sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
01700     sprintf(repl, "%d", year);

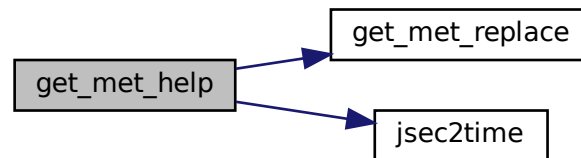
```

```

01701  get_met_replace(filename, "YYYY", repl);
01702  sprintf(repl, "%02d", mon);
01703  get_met_replace(filename, "MM", repl);
01704  sprintf(repl, "%02d", day);
01705  get_met_replace(filename, "DD", repl);
01706  sprintf(repl, "%02d", hour);
01707  get_met_replace(filename, "HH", repl);
01708  }

```

Here is the call graph for this function:



**5.21.2.10 `get_met_replace()`** `void get_met_replace (`  
     `char * orig,`  
     `char * search,`  
     `char * repl )`

Replace template strings in filename.

Definition at line 1712 of file `libtrac.c`.

```

01715  {
01716
01717  char buffer[LEN], *ch;
01718
01719  /* Iterate... */
01720  for (int i = 0; i < 3; i++) {
01721
01722  /* Replace sub-string... */
01723  if (!(ch = strstr(orig, search)))
01724  return;
01725  strncpy(buffer, orig, (size_t)(ch - orig));
01726  buffer[ch - orig] = 0;
01727  sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
01728  orig[0] = 0;
01729  strcpy(orig, buffer);
01730  }
01731 }

```

**5.21.2.11 `intpol_met_space_3d()`** `void intpol_met_space_3d (`  
     `met_t * met,`  
     `float array[EX][EY][EP],`  
     `double p,`  
     `double lon,`  
     `double lat,`  
     `double * var,`  
     `int * ci,`

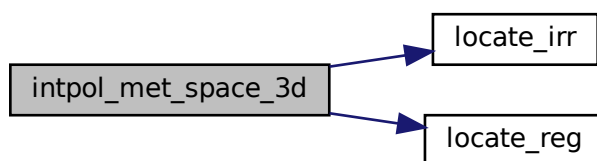
```
double * cw,
int init )
```

Spatial interpolation of meteorological data.

Definition at line 1735 of file libtrac.c.

```
01744     {
01745
01746     /* Check longitude... */
01747     if (met->lon[met->nx - 1] > 180 && lon < 0)
01748         lon += 360;
01749
01750     /* Get interpolation indices and weights... */
01751     if (init) {
01752         ci[0] = locate_irr(met->p, met->np, p);
01753         ci[1] = locate_reg(met->lon, met->nx, lon);
01754         ci[2] = locate_reg(met->lat, met->ny, lat);
01755         cw[0] = (met->p[ci[0] + 1] - p)
01756             / (met->p[ci[0] + 1] - met->p[ci[0]]);
01757         cw[1] = (met->lon[ci[1] + 1] - lon)
01758             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01759         cw[2] = (met->lat[ci[2] + 1] - lat)
01760             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01761     }
01762
01763     /* Interpolate vertically... */
01764     double aux00 =
01765         cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
01766         + array[ci[1]][ci[2]][ci[0] + 1];
01767     double aux01 =
01768         cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
01769             array[ci[1]][ci[2] + 1][ci[0] + 1])
01770         + array[ci[1]][ci[2] + 1][ci[0] + 1];
01771     double aux10 =
01772         cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
01773             array[ci[1] + 1][ci[2]][ci[0] + 1])
01774         + array[ci[1] + 1][ci[2]][ci[0] + 1];
01775     double aux11 =
01776         cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
01777             array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
01778         + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];
01779
01780     /* Interpolate horizontally... */
01781     aux00 = cw[2] * (aux00 - aux01) + aux01;
01782     aux11 = cw[2] * (aux10 - aux11) + aux11;
01783     *var = cw[1] * (aux00 - aux11) + aux11;
01784 }
```

Here is the call graph for this function:



**5.21.2.12 intpol\_met\_space\_2d()** void intpol\_met\_space\_2d (  
     met\_t \* met,  
     float array[EX][EY],  
     double lon,



```

double lat,
double * var,
int * ci,
double * cw,
int init )

```

Spatial interpolation of meteorological data.

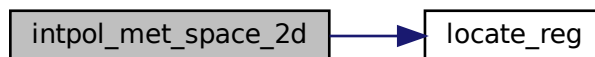
Definition at line 1789 of file libtrac.c.

```

01797     {
01798
01799     /* Check longitude... */
01800     if (met->lon[met->nx - 1] > 180 && lon < 0)
01801         lon += 360;
01802
01803     /* Get interpolation indices and weights... */
01804     if (init) {
01805         ci[1] = locate_reg(met->lon, met->nx, lon);
01806         ci[2] = locate_reg(met->lat, met->ny, lat);
01807         cw[1] = (met->lon[ci[1] + 1] - lon)
01808             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01809         cw[2] = (met->lat[ci[2] + 1] - lat)
01810             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01811     }
01812
01813     /* Set variables... */
01814     double aux00 = array[ci[1]][ci[2]];
01815     double aux01 = array[ci[1]][ci[2] + 1];
01816     double aux10 = array[ci[1] + 1][ci[2]];
01817     double aux11 = array[ci[1] + 1][ci[2] + 1];
01818
01819     /* Interpolate horizontally... */
01820     if (isfinite(aux00) && isfinite(aux01))
01821         aux00 = cw[2] * (aux00 - aux01) + aux01;
01822     else if (cw[2] < 0.5)
01823         aux00 = aux01;
01824     if (isfinite(aux10) && isfinite(aux11))
01825         aux11 = cw[2] * (aux10 - aux11) + aux11;
01826     else if (cw[2] > 0.5)
01827         aux11 = aux10;
01828     if (isfinite(aux00) && isfinite(aux11))
01829         *var = cw[1] * (aux00 - aux11) + aux11;
01830     else {
01831         if (cw[1] > 0.5)
01832             *var = aux00;
01833         else
01834             *var = aux11;
01835     }
01836 }

```

Here is the call graph for this function:



**5.21.2.13 intpol\_met\_time\_3d()** void intpol\_met\_time\_3d (

```

    met_t * met0,
    float array0[EX][EY][EP],
    met_t * met1,
    float array1[EX][EY][EP],

```

```

double ts,
double p,
double lon,
double lat,
double * var,
int * ci,
double * cw,
int init )

```

Temporal interpolation of meteorological data.

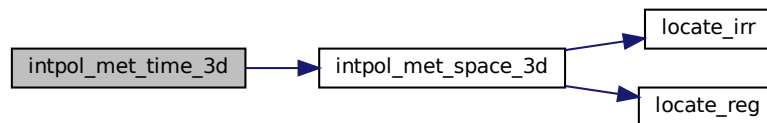
Definition at line 1840 of file libtrac.c.

```

1852     {
1853
1854     double var0, var1, wt;
1855
1856     /* Spatial interpolation... */
1857     intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
1858     intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, init);
1859
1860     /* Get weighting factor... */
1861     wt = (met1->time - ts) / (met1->time - met0->time);
1862
1863     /* Interpolate... */
1864     *var = wt * (var0 - var1) + var1;
1865 }

```

Here is the call graph for this function:



#### 5.21.2.14 intpol\_met\_time\_2d() void intpol\_met\_time\_2d (

```

    met_t * met0,
    float array0[EX][EY],
    met_t * met1,
    float array1[EX][EY],
    double ts,
    double lon,
    double lat,
    double * var,
    int * ci,
    double * cw,
    int init )

```

Temporal interpolation of meteorological data.

Definition at line 1869 of file libtrac.c.

```

1880     {
1881
1882     double var0, var1, wt;
1883
1884     /* Spatial interpolation... */

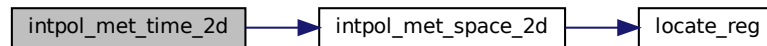
```

```

01885  intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
01886  intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, init);
01887
01888  /* Get weighting factor... */
01889  wt = (met1->time - ts) / (met1->time - met0->time);
01890
01891  /* Interpolate... */
01892  *var = wt * (var0 - var1) + var1;
01893 }

```

Here is the call graph for this function:



**5.21.2.15 jsec2time()** 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 1897 of file libtrac.c.

```

01905  {
01906
01907  struct tm t0, *t1;
01908
01909  t0.tm_year = 100;
01910  t0.tm_mon = 0;
01911  t0.tm_mday = 1;
01912  t0.tm_hour = 0;
01913  t0.tm_min = 0;
01914  t0.tm_sec = 0;
01915
01916  time_t jsec0 = (time_t) jsec + timegm(&t0);
01917  t1 = gmtime(&jsec0);
01918
01919  *year = t1->tm_year + 1900;
01920  *mon = t1->tm_mon + 1;
01921  *day = t1->tm_mday;
01922  *hour = t1->tm_hour;
01923  *min = t1->tm_min;
01924  *sec = t1->tm_sec;
01925  *remain = jsec - floor(jsec);
01926 }

```

**5.21.2.16 lapse\_rate()** double lapse\_rate (  
double t,  
double h2o )

Calculate moist adiabatic lapse rate.

Definition at line 1930 of file libtrac.c.

```
01932     {
01933
01934     /*
01935      Calculate moist adiabatic lapse rate [K/km] from temperature [K]
01936      and water vapor volume mixing ratio [1].
01937
01938      Reference: https://en.wikipedia.org/wiki/Lapse\_rate
01939     */
01940
01941     const double a = RA * SQR(t), r = SH(h2o) / (1. - SH(h2o));
01942
01943     return 1e3 * G0 * (a + LV * r * t) / (CPD * a + SQR(LV) * r * EPS);
01944 }
```

**5.21.2.17 locate\_irr()** int locate\_irr (  
double \* xx,  
int n,  
double x )

Find array index for irregular grid.

Definition at line 1948 of file libtrac.c.

```
01951     {
01952
01953     int ilo = 0;
01954     int ihi = n - 1;
01955     int i = (ihi + ilo) » 1;
01956
01957     if (xx[i] < xx[i + 1])
01958         while (ihi > ilo + 1) {
01959             i = (ihi + ilo) » 1;
01960             if (xx[i] > x)
01961                 ihi = i;
01962             else
01963                 ilo = i;
01964         } else
01965         while (ihi > ilo + 1) {
01966             i = (ihi + ilo) » 1;
01967             if (xx[i] <= x)
01968                 ihi = i;
01969             else
01970                 ilo = i;
01971         }
01972
01973     return ilo;
01974 }
```

**5.21.2.18 locate\_reg()** int locate\_reg (  
double \* xx,  
int n,  
double x )

Find array index for regular grid.

Definition at line 1978 of file libtrac.c.

```
01981     {
01982
01983     /* Calculate index... */
01984     int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
```

```

01985
01986  /* Check range... */
01987  if (i < 0)
01988      i = 0;
01989  else if (i >= n - 2)
01990      i = n - 2;
01991
01992  return i;
01993 }

```

**5.21.2.19 nat\_temperature()** double nat\_temperature (  
     double *p*,  
     double *h2o*,  
     double *hno3* )

Calculate NAT existence temperature.

Definition at line 1997 of file [libtrac.c](#).

```

02000      {
02001
02002      double p_hno3 = hno3 * p / 1.333224;
02003      double p_h2o = h2o * p / 1.333224;
02004      double a = 0.009179 - 0.00088 * log10(p_h2o);
02005      double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
02006      double c = -11397.0 / a;
02007      double tnat = (-b + sqrt(b * b - 4. * c)) / 2.;
02008      double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
02009      if (x2 > 0)
02010          tnat = x2;
02011
02012      return tnat;
02013 }

```

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

Read atmospheric data.

Definition at line 2017 of file [libtrac.c](#).

```

02020      {
02021
02022      FILE *in;
02023
02024      char line[LEN], *tok;
02025
02026      double t0;
02027
02028      int dimid, ip, iq, ncid, varid;
02029
02030      size_t nparts;
02031
02032      /* Set timer... */
02033      SELECT_TIMER("READ_ATM", NVTX_READ);
02034
02035      /* Init... */
02036      atm->np = 0;
02037
02038      /* Write info... */
02039      printf("Read atmospheric data: %s\n", filename);
02040
02041      /* Read ASCII data... */
02042      if (ctl->atm_type == 0) {
02043
02044          /* Open file... */
02045          if (!(in = fopen(filename, "r"))) {
02046              WARN("File not found!");
02047              return 0;

```

```

02048     }
02049
02050     /* Read line... */
02051     while (fgets(line, LEN, in)) {
02052
02053         /* Read data... */
02054         TOK(line, tok, "%lg", atm->time[atm->np]);
02055         TOK(NULL, tok, "%lg", atm->p[atm->np]);
02056         TOK(NULL, tok, "%lg", atm->lon[atm->np]);
02057         TOK(NULL, tok, "%lg", atm->lat[atm->np]);
02058         for (iq = 0; iq < ctl->nq; iq++)
02059             TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
02060
02061         /* Convert altitude to pressure... */
02062         atm->p[atm->np] = P(atm->p[atm->np]);
02063
02064         /* Increment data point counter... */
02065         if ((++atm->np) > NP)
02066             ERRMSG("Too many data points!");
02067     }
02068
02069     /* Close file... */
02070     fclose(in);
02071 }
02072
02073 /* Read binary data... */
02074 else if (ctl->atm_type == 1) {
02075
02076     /* Open file... */
02077     if (!(in = fopen(filename, "r")))
02078         return 0;
02079
02080     /* Read data... */
02081     FREAD(&atm->np, int,
02082          1,
02083          in);
02084     FREAD(atm->time, double,
02085          (size_t) atm->np,
02086          in);
02087     FREAD(atm->p, double,
02088          (size_t) atm->np,
02089          in);
02090     FREAD(atm->lon, double,
02091          (size_t) atm->np,
02092          in);
02093     FREAD(atm->lat, double,
02094          (size_t) atm->np,
02095          in);
02096     for (iq = 0; iq < ctl->nq; iq++)
02097         FREAD(atm->q[iq], double,
02098              (size_t) atm->np,
02099              in);
02100
02101     /* Close file... */
02102     fclose(in);
02103 }
02104
02105 /* Read netCDF data... */
02106 else if (ctl->atm_type == 2) {
02107
02108     /* Open file... */
02109     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02110         return 0;
02111
02112     /* Get dimensions... */
02113     NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
02114     NC(nc_inq_dimlen(ncid, dimid, &nparts));
02115     atm->np = (int) nparts;
02116     if (atm->np > NP)
02117         ERRMSG("Too many particles!");
02118
02119     /* Get time... */
02120     NC(nc_inq_varid(ncid, "time", &varid));
02121     NC(nc_get_var_double(ncid, varid, &t0));
02122     for (ip = 0; ip < atm->np; ip++)
02123         atm->time[ip] = t0;
02124
02125     /* Read geolocations... */
02126     NC(nc_inq_varid(ncid, "PRESS", &varid));
02127     NC(nc_get_var_double(ncid, varid, atm->p));
02128     NC(nc_inq_varid(ncid, "LON", &varid));
02129     NC(nc_get_var_double(ncid, varid, atm->lon));
02130     NC(nc_inq_varid(ncid, "LAT", &varid));
02131     NC(nc_get_var_double(ncid, varid, atm->lat));
02132
02133     /* Read variables... */
02134     if (ctl->qnt_p >= 0)

```

```

02135     if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
02136         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
02137     if (ctl->qnt_t >= 0)
02138         if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
02139             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
02140     if (ctl->qnt_u >= 0)
02141         if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
02142             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
02143     if (ctl->qnt_v >= 0)
02144         if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
02145             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
02146     if (ctl->qnt_w >= 0)
02147         if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
02148             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
02149     if (ctl->qnt_h2o >= 0)
02150         if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
02151             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
02152     if (ctl->qnt_o3 >= 0)
02153         if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
02154             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
02155     if (ctl->qnt_theta >= 0)
02156         if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
02157             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
02158     if (ctl->qnt_pv >= 0)
02159         if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
02160             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
02161
02162     /* Check data... */
02163     for (ip = 0; ip < atm->np; ip++)
02164         if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
02165             || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
02166             || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
02167             || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
02168             || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
02169         atm->time[ip] = GSL_NAN;
02170         atm->p[ip] = GSL_NAN;
02171         atm->lon[ip] = GSL_NAN;
02172         atm->lat[ip] = GSL_NAN;
02173         for (iq = 0; iq < ctl->nq; iq++)
02174             atm->q[iq][ip] = GSL_NAN;
02175     } else {
02176         if (ctl->qnt_h2o >= 0)
02177             atm->q[ctl->qnt_h2o][ip] *= 1.608;
02178         if (ctl->qnt_pv >= 0)
02179             atm->q[ctl->qnt_pv][ip] *= 1e6;
02180         if (atm->lon[ip] > 180)
02181             atm->lon[ip] -= 360;
02182     }
02183
02184     /* Close file... */
02185     NC(nc_close(ncid));
02186 }
02187
02188 /* Error... */
02189 else
02190     ERRMSG("Atmospheric data type not supported!");
02191
02192 /* Check number of points... */
02193 if (atm->np < 1)
02194     ERRMSG("Can not read any data!");
02195
02196 /* Return success... */
02197 return 1;
02198 }

```

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

Read control parameters.

Definition at line 2202 of file libtrac.c.

```

02206     {
02207
02208     /* Set timer... */

```

```

02209 SELECT_TIMER("READ_CTL", NVTX_READ);
02210
02211 /* Write info... */
02212 printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
02213        "(executable: %s | compiled: %s, %s)\n\n",
02214        argv[0], __DATE__, __TIME__);
02215
02216 /* Initialize quantity indices... */
02217 ctl->qnt_ens = -1;
02218 ctl->qnt_m = -1;
02219 ctl->qnt_r = -1;
02220 ctl->qnt_rho = -1;
02221 ctl->qnt_ps = -1;
02222 ctl->qnt_ts = -1;
02223 ctl->qnt_zs = -1;
02224 ctl->qnt_us = -1;
02225 ctl->qnt_vs = -1;
02226 ctl->qnt_pt = -1;
02227 ctl->qnt_tt = -1;
02228 ctl->qnt_zt = -1;
02229 ctl->qnt_h2ot = -1;
02230 ctl->qnt_z = -1;
02231 ctl->qnt_p = -1;
02232 ctl->qnt_t = -1;
02233 ctl->qnt_u = -1;
02234 ctl->qnt_v = -1;
02235 ctl->qnt_w = -1;
02236 ctl->qnt_h2o = -1;
02237 ctl->qnt_o3 = -1;
02238 ctl->qnt_lwc = -1;
02239 ctl->qnt_iwc = -1;
02240 ctl->qnt_pc = -1;
02241 ctl->qnt_cl = -1;
02242 ctl->qnt_plcl = -1;
02243 ctl->qnt_plfc = -1;
02244 ctl->qnt_pel = -1;
02245 ctl->qnt_cape = -1;
02246 ctl->qnt_hno3 = -1;
02247 ctl->qnt_oh = -1;
02248 ctl->qnt_psat = -1;
02249 ctl->qnt_psize = -1;
02250 ctl->qnt_pw = -1;
02251 ctl->qnt_sh = -1;
02252 ctl->qnt_rh = -1;
02253 ctl->qnt_rhice = -1;
02254 ctl->qnt_theta = -1;
02255 ctl->qnt_tvirt = -1;
02256 ctl->qnt_lapse = -1;
02257 ctl->qnt_vh = -1;
02258 ctl->qnt_vz = -1;
02259 ctl->qnt_pv = -1;
02260 ctl->qnt_tdew = -1;
02261 ctl->qnt_tice = -1;
02262 ctl->qnt_tsts = -1;
02263 ctl->qnt_tnat = -1;
02264 ctl->qnt_stat = -1;
02265
02266 /* Read quantities... */
02267 ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
02268 if (ctl->nq > NQ)
02269     ERRMSG("Too many quantities!");
02270 for (int iq = 0; iq < ctl->nq; iq++) {
02271
02272     /* Read quantity name and format... */
02273     scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
02274     scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
02275             ctl->qnt_format[iq]);
02276
02277     /* Try to identify quantity... */
02278     if (strcasecmp(ctl->qnt_name[iq], "ens") == 0) {
02279         ctl->qnt_ens = iq;
02280         sprintf(ctl->qnt_unit[iq], "-");
02281     } else if (strcasecmp(ctl->qnt_name[iq], "m") == 0) {
02282         ctl->qnt_m = iq;
02283         sprintf(ctl->qnt_unit[iq], "kg");
02284     } else if (strcasecmp(ctl->qnt_name[iq], "r") == 0) {
02285         ctl->qnt_r = iq;
02286         sprintf(ctl->qnt_unit[iq], "m");
02287     } else if (strcasecmp(ctl->qnt_name[iq], "rho") == 0) {
02288         ctl->qnt_rho = iq;
02289         sprintf(ctl->qnt_unit[iq], "kg/m^3");
02290     } else if (strcasecmp(ctl->qnt_name[iq], "ps") == 0) {
02291         ctl->qnt_ps = iq;
02292         sprintf(ctl->qnt_unit[iq], "hPa");
02293     } else if (strcasecmp(ctl->qnt_name[iq], "pt") == 0) {
02294         ctl->qnt_pt = iq;
02295         sprintf(ctl->qnt_unit[iq], "hPa");
02296     }
02297 }

```



```

02296     } else if (strcasecmp(ctl->qnt_name[iq], "tt") == 0) {
02297         ctl->qnt_tt = iq;
02298         sprintf(ctl->qnt_unit[iq], "K");
02299     } else if (strcasecmp(ctl->qnt_name[iq], "zt") == 0) {
02300         ctl->qnt_zt = iq;
02301         sprintf(ctl->qnt_unit[iq], "km");
02302     } else if (strcasecmp(ctl->qnt_name[iq], "h2ot") == 0) {
02303         ctl->qnt_h2ot = iq;
02304         sprintf(ctl->qnt_unit[iq], "ppv");
02305     } else if (strcasecmp(ctl->qnt_name[iq], "z") == 0) {
02306         ctl->qnt_z = iq;
02307         sprintf(ctl->qnt_unit[iq], "km");
02308     } else if (strcasecmp(ctl->qnt_name[iq], "p") == 0) {
02309         ctl->qnt_p = iq;
02310         sprintf(ctl->qnt_unit[iq], "hPa");
02311     } else if (strcasecmp(ctl->qnt_name[iq], "t") == 0) {
02312         ctl->qnt_t = iq;
02313         sprintf(ctl->qnt_unit[iq], "K");
02314     } else if (strcasecmp(ctl->qnt_name[iq], "u") == 0) {
02315         ctl->qnt_u = iq;
02316         sprintf(ctl->qnt_unit[iq], "m/s");
02317     } else if (strcasecmp(ctl->qnt_name[iq], "v") == 0) {
02318         ctl->qnt_v = iq;
02319         sprintf(ctl->qnt_unit[iq], "m/s");
02320     } else if (strcasecmp(ctl->qnt_name[iq], "w") == 0) {
02321         ctl->qnt_w = iq;
02322         sprintf(ctl->qnt_unit[iq], "hPa/s");
02323     } else if (strcasecmp(ctl->qnt_name[iq], "h2o") == 0) {
02324         ctl->qnt_h2o = iq;
02325         sprintf(ctl->qnt_unit[iq], "ppv");
02326     } else if (strcasecmp(ctl->qnt_name[iq], "o3") == 0) {
02327         ctl->qnt_o3 = iq;
02328         sprintf(ctl->qnt_unit[iq], "ppv");
02329     } else if (strcasecmp(ctl->qnt_name[iq], "lwc") == 0) {
02330         ctl->qnt_lwc = iq;
02331         sprintf(ctl->qnt_unit[iq], "kg/kg");
02332     } else if (strcasecmp(ctl->qnt_name[iq], "iwc") == 0) {
02333         ctl->qnt_iwc = iq;
02334         sprintf(ctl->qnt_unit[iq], "kg/kg");
02335     } else if (strcasecmp(ctl->qnt_name[iq], "pc") == 0) {
02336         ctl->qnt_pc = iq;
02337         sprintf(ctl->qnt_unit[iq], "hPa");
02338     } else if (strcasecmp(ctl->qnt_name[iq], "cl") == 0) {
02339         ctl->qnt_cl = iq;
02340         sprintf(ctl->qnt_unit[iq], "kg/m^2");
02341     } else if (strcasecmp(ctl->qnt_name[iq], "plcl") == 0) {
02342         ctl->qnt_plcl = iq;
02343         sprintf(ctl->qnt_unit[iq], "hPa");
02344     } else if (strcasecmp(ctl->qnt_name[iq], "plfc") == 0) {
02345         ctl->qnt_plfc = iq;
02346         sprintf(ctl->qnt_unit[iq], "hPa");
02347     } else if (strcasecmp(ctl->qnt_name[iq], "pel") == 0) {
02348         ctl->qnt_pel = iq;
02349         sprintf(ctl->qnt_unit[iq], "hPa");
02350     } else if (strcasecmp(ctl->qnt_name[iq], "cape") == 0) {
02351         ctl->qnt_cape = iq;
02352         sprintf(ctl->qnt_unit[iq], "J/kg");
02353     } else if (strcasecmp(ctl->qnt_name[iq], "hno3") == 0) {
02354         ctl->qnt_hno3 = iq;
02355         sprintf(ctl->qnt_unit[iq], "ppv");
02356     } else if (strcasecmp(ctl->qnt_name[iq], "oh") == 0) {
02357         ctl->qnt_oh = iq;
02358         sprintf(ctl->qnt_unit[iq], "molec/cm^3");
02359     } else if (strcasecmp(ctl->qnt_name[iq], "psat") == 0) {
02360         ctl->qnt_psat = iq;
02361         sprintf(ctl->qnt_unit[iq], "hPa");
02362     } else if (strcasecmp(ctl->qnt_name[iq], "psice") == 0) {
02363         ctl->qnt_psice = iq;
02364         sprintf(ctl->qnt_unit[iq], "hPa");
02365     } else if (strcasecmp(ctl->qnt_name[iq], "pw") == 0) {
02366         ctl->qnt_pw = iq;
02367         sprintf(ctl->qnt_unit[iq], "hPa");
02368     } else if (strcasecmp(ctl->qnt_name[iq], "sh") == 0) {
02369         ctl->qnt_sh = iq;
02370         sprintf(ctl->qnt_unit[iq], "kg/kg");
02371     } else if (strcasecmp(ctl->qnt_name[iq], "rh") == 0) {
02372         ctl->qnt_rh = iq;
02373         sprintf(ctl->qnt_unit[iq], "%");
02374     } else if (strcasecmp(ctl->qnt_name[iq], "rhice") == 0) {
02375         ctl->qnt_rhice = iq;
02376         sprintf(ctl->qnt_unit[iq], "%");
02377     } else if (strcasecmp(ctl->qnt_name[iq], "theta") == 0) {
02378         ctl->qnt_theta = iq;
02379         sprintf(ctl->qnt_unit[iq], "K");
02380     } else if (strcasecmp(ctl->qnt_name[iq], "tvirt") == 0) {
02381         ctl->qnt_tvirt = iq;
02382         sprintf(ctl->qnt_unit[iq], "K");

```

```

02383     } else if (strcasecmp(ctl->qnt_name[iq], "lapse") == 0) {
02384         ctl->qnt_lapse = iq;
02385         sprintf(ctl->qnt_unit[iq], "K/km");
02386     } else if (strcasecmp(ctl->qnt_name[iq], "vh") == 0) {
02387         ctl->qnt_vh = iq;
02388         sprintf(ctl->qnt_unit[iq], "m/s");
02389     } else if (strcasecmp(ctl->qnt_name[iq], "vz") == 0) {
02390         ctl->qnt_vz = iq;
02391         sprintf(ctl->qnt_unit[iq], "m/s");
02392     } else if (strcasecmp(ctl->qnt_name[iq], "pv") == 0) {
02393         ctl->qnt_pv = iq;
02394         sprintf(ctl->qnt_unit[iq], "PVU");
02395     } else if (strcasecmp(ctl->qnt_name[iq], "tdew") == 0) {
02396         ctl->qnt_tdew = iq;
02397         sprintf(ctl->qnt_unit[iq], "K");
02398     } else if (strcasecmp(ctl->qnt_name[iq], "tice") == 0) {
02399         ctl->qnt_tice = iq;
02400         sprintf(ctl->qnt_unit[iq], "K");
02401     } else if (strcasecmp(ctl->qnt_name[iq], "tsts") == 0) {
02402         ctl->qnt_tsts = iq;
02403         sprintf(ctl->qnt_unit[iq], "K");
02404     } else if (strcasecmp(ctl->qnt_name[iq], "tnat") == 0) {
02405         ctl->qnt_tnat = iq;
02406         sprintf(ctl->qnt_unit[iq], "K");
02407     } else if (strcasecmp(ctl->qnt_name[iq], "stat") == 0) {
02408         ctl->qnt_stat = iq;
02409         sprintf(ctl->qnt_unit[iq], "-");
02410     } else
02411         scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
02412 }
02413
02414 /* Time steps of simulation... */
02415 ctl->direction =
02416     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
02417 if (ctl->direction != -1 && ctl->direction != 1)
02418     ERRMSG("Set DIRECTION to -1 or 1!");
02419 ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
02420 ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "180", NULL);
02421
02422 /* Meteorological data... */
02423 scan_ctl(filename, argc, argv, "METBASE", -1, "-", ctl->metbase);
02424 ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
02425 ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
02426 ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
02427 ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
02428 if (ctl->met_dx < 1 || ctl->met_dy < 1 || ctl->met_dp < 1)
02429     ERRMSG("MET_DX, MET_DY, and MET_DP need to be greater than zero!");
02430 ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
02431 ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
02432 ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
02433 if (ctl->met_sx < 1 || ctl->met_sy < 1 || ctl->met_sp < 1)
02434     ERRMSG("MET_SX, MET_SY, and MET_SP need to be greater than zero!");
02435 ctl->met_detrend =
02436     scan_ctl(filename, argc, argv, "MET_DETREND", -1, "-999", NULL);
02437 ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
02438 if (ctl->met_np > EP)
02439     ERRMSG("Too many levels!");
02440 for (int ip = 0; ip < ctl->met_np; ip++)
02441     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
02442 ctl->met_geopot_sx
02443     = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SX", -1, "6", NULL);
02444 ctl->met_geopot_sy
02445     = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SY", -1, "4", NULL);
02446 if (ctl->met_geopot_sx < 1 || ctl->met_geopot_sy < 1)
02447     ERRMSG("MET_GEOPOT_SX and MET_GEOPOT_SY need to be greater than zero!");
02448 ctl->met_tropo =
02449     (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
02450 ctl->met_dt_out =
02451     scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
02452
02453 /* Isosurface parameters... */
02454 ctl->isosurf =
02455     (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
02456 scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
02457
02458 /* Diffusion parameters... */
02459 ctl->turb_dx_trop =
02460     scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
02461 ctl->turb_dx_strat =
02462     scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
02463 ctl->turb_dz_trop =
02464     scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
02465 ctl->turb_dz_strat =
02466     scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
02467 ctl->turb_mesox =
02468     scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
02469 ctl->turb_mesoz =

```

```

02470     scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
02471
02472     /* Convection... */
02473     ctl->conv_cape =
02474         scan_ctl(filename, argc, argv, "CONV_CAPE", -1, "-999", NULL);
02475
02476     /* Species parameters... */
02477     scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
02478     if (strcasecmp(ctl->species, "SO2") == 0) {
02479         ctl->molmass = 64.066;
02480         ctl->oh_chem[0] = 3.3e-31; /* (JPL Publication 15-10) */
02481         ctl->oh_chem[1] = 4.3; /* (JPL Publication 15-10) */
02482         ctl->oh_chem[2] = 1.6e-12; /* (JPL Publication 15-10) */
02483         ctl->oh_chem[3] = 0.0; /* (JPL Publication 15-10) */
02484         ctl->wet_depo[2] = 1.3e-2; /* (Sander, 2015) */
02485         ctl->wet_depo[3] = 2900.0; /* (Sander, 2015) */
02486         ctl->wet_depo[6] = 1.3e-2; /* (Sander, 2015) */
02487         ctl->wet_depo[7] = 2900.0; /* (Sander, 2015) */
02488     } else {
02489         ctl->molmass =
02490             scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
02491         ctl->tdec_trop =
02492             scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
02493         ctl->tdec_strat =
02494             scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
02495         for (int ip = 0; ip < 4; ip++)
02496             ctl->oh_chem[ip] =
02497                 scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
02498         for (int ip = 0; ip < 1; ip++)
02499             ctl->dry_depo[ip] =
02500                 scan_ctl(filename, argc, argv, "DRY_DEPO", ip, "0", NULL);
02501         for (int ip = 0; ip < 8; ip++)
02502             ctl->wet_depo[ip] =
02503                 scan_ctl(filename, argc, argv, "WET_DEPO", ip, "0", NULL);
02504     }
02505
02506     /* PSC analysis... */
02507     ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
02508     ctl->psc_hno3 =
02509         scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
02510
02511     /* Output of atmospheric data... */
02512     scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->atm_basename);
02513     scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
02514     ctl->atm_dt_out =
02515         scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
02516     ctl->atm_filter =
02517         (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
02518     ctl->atm_stride =
02519         (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
02520     ctl->atm_type =
02521         (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
02522
02523     /* Output of CSI data... */
02524     scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->csi_basename);
02525     ctl->csi_dt_out =
02526         scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
02527     scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->csi_obsfile);
02528     ctl->csi_obsmin =
02529         scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
02530     ctl->csi_modmin =
02531         scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
02532     ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
02533     ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
02534     ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
02535     ctl->csi_lon0 =
02536         scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
02537     ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
02538     ctl->csi_nx =
02539         (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
02540     ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
02541     ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
02542     ctl->csi_ny =
02543         (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
02544
02545     /* Output of ensemble data... */
02546     scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->ens_basename);
02547
02548     /* Output of grid data... */
02549     scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
02550         ctl->grid_basename);
02551     scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->grid_gpfile);
02552     ctl->grid_dt_out =
02553         scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
02554     ctl->grid_sparse =
02555         (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
02556     ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);

```

```

02557     ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
02558     ctl->grid_nz =
02559         (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
02560     ctl->grid_lon0 =
02561         scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
02562     ctl->grid_lon1 =
02563         scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
02564     ctl->grid_nx =
02565         (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
02566     ctl->grid_lat0 =
02567         scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
02568     ctl->grid_lat1 =
02569         scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
02570     ctl->grid_ny =
02571         (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
02572
02573     /* Output of profile data... */
02574     scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
02575         ctl->prof_basename);
02576     scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->prof_obsfile);
02577     ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
02578     ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
02579     ctl->prof_nz =
02580         (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
02581     ctl->prof_lon0 =
02582         scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
02583     ctl->prof_lon1 =
02584         scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
02585     ctl->prof_nx =
02586         (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
02587     ctl->prof_lat0 =
02588         scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
02589     ctl->prof_lat1 =
02590         scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
02591     ctl->prof_ny =
02592         (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
02593
02594     /* Output of sample data... */
02595     scan_ctl(filename, argc, argv, "SAMPLE_BASENAME", -1, "-",
02596         ctl->sample_basename);
02597     scan_ctl(filename, argc, argv, "SAMPLE_OBSFILE", -1, "-",
02598         ctl->sample_obsfile);
02599     ctl->sample_dx =
02600         scan_ctl(filename, argc, argv, "SAMPLE_DX", -1, "50", NULL);
02601     ctl->sample_dz =
02602         scan_ctl(filename, argc, argv, "SAMPLE_DZ", -1, "-999", NULL);
02603
02604     /* Output of station data... */
02605     scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
02606         ctl->stat_basename);
02607     ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
02608     ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
02609     ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
02610 }

```

Here is the call graph for this function:



**5.21.2.22 read\_met()** int read\_met (

```

    ctl_t * ctl,
    char * filename,
    met_t * met )

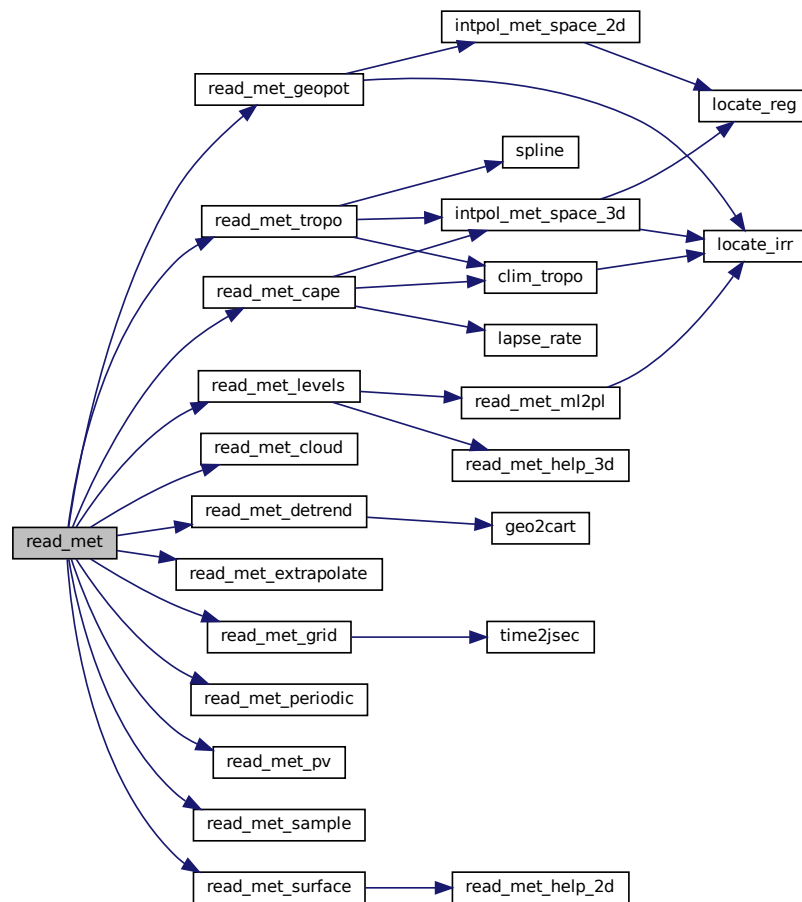
```

Read meteorological data file.

Definition at line 2614 of file libtrac.c.

```
02617         {
02618
02619     int ncid;
02620
02621     /* Write info... */
02622     printf("Read meteorological data: %s\n", filename);
02623
02624     /* Open netCDF file... */
02625     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
02626         WARN("File not found!");
02627         return 0;
02628     }
02629
02630     /* Read coordinates of meteorological data... */
02631     read_met_grid(filename, ncid, ctl, met);
02632
02633     /* Read meteo data on vertical levels... */
02634     read_met_levels(ncid, ctl, met);
02635
02636     /* Extrapolate data for lower boundary... */
02637     read_met_extrapolate(met);
02638
02639     /* Read surface data... */
02640     read_met_surface(ncid, met);
02641
02642     /* Create periodic boundary conditions... */
02643     read_met_periodic(met);
02644
02645     /* Downsampling... */
02646     read_met_sample(ctl, met);
02647
02648     /* Calculate geopotential heights... */
02649     read_met_geopot(ctl, met);
02650
02651     /* Calculate potential vorticity... */
02652     read_met_pv(met);
02653
02654     /* Calculate tropopause data... */
02655     read_met_tropo(ctl, met);
02656
02657     /* Calculate cloud properties... */
02658     read_met_cloud(met);
02659
02660     /* Calculate convective available potential energy... */
02661     read_met_cape(met);
02662
02663     /* Detrending... */
02664     read_met_detrend(ctl, met);
02665
02666     /* Close file... */
02667     NC(nc_close(ncid));
02668
02669     /* Return success... */
02670     return 1;
02671 }
```

Here is the call graph for this function:



**5.21.2.23 read\_met\_cape()** void read\_met\_cape (  
     met\_t \* met )

Calculate convective available potential energy.

Definition at line 2675 of file libtrac.c.

```

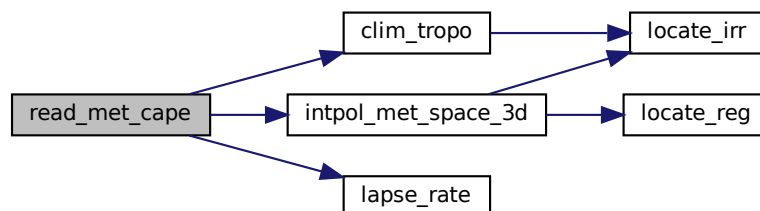
02676     {
02677
02678     /* Set timer... */
02679     SELECT_TIMER("READ_MET_CAPE", NVTX_READ);
02680
02681     /* Vertical spacing (about 100 m)... */
02682     const double pfac = 1.01439, dz0 = RI / MA / G0 * log(pfac);
02683
02684     /* Loop over columns... */
02685     #pragma omp parallel for default(shared)
02686     for (int ix = 0; ix < met->nx; ix++)
02687         for (int iy = 0; iy < met->ny; iy++) {
02688
02689         /* Get potential temperature and water vapor vmr at lowest 50 hPa... */
02690         int n = 0;
02691         double h2o = 0, t, theta = 0;
02692         double pbot = GSL_MIN(met->ps[ix][iy], met->p[0]);
02693         double ptop = pbot - 50.;
  
```

```

02694     for (int ip = 0; ip < met->np; ip++) {
02695         if (met->p[ip] <= pbot) {
02696             theta += THETA(met->p[ip], met->t[ix][iy][ip]);
02697             h2o += met->h2o[ix][iy][ip];
02698             n++;
02699         }
02700         if (met->p[ip] < ptop && n > 0)
02701             break;
02702     }
02703     theta /= n;
02704     h2o /= n;
02705
02706     /* Cannot compute anything if water vapor is missing... */
02707     met->plcl[ix][iy] = GSL_NAN;
02708     met->plfc[ix][iy] = GSL_NAN;
02709     met->pel[ix][iy] = GSL_NAN;
02710     met->cape[ix][iy] = GSL_NAN;
02711     if (h2o <= 0)
02712         continue;
02713
02714     /* Find lifted condensation level (LCL)... */
02715     ptop = P(20.);
02716     pbot = met->ps[ix][iy];
02717     do {
02718         met->plcl[ix][iy] = (float) (0.5 * (pbot + ptop));
02719         t = theta / pow(1000. / met->plcl[ix][iy], 0.286);
02720         if (RH(met->plcl[ix][iy], t, h2o) > 100.)
02721             ptop = met->plcl[ix][iy];
02722         else
02723             pbot = met->plcl[ix][iy];
02724     } while (pbot - ptop > 0.1);
02725
02726     /* Calculate level of free convection (LFC), equilibrium level (EL),
02727        and convective available potential energy (CAPE)... */
02728     double dcape = 0, dcape_old, dz, h2o_env, p = met->plcl[ix][iy],
02729         psat, t_env;
02730     ptop = 0.75 * clim_tropo(met->time, met->lat[iy]);
02731     met->cape[ix][iy] = 0;
02732     do {
02733         dz = dz0 * TVIRT(t, h2o);
02734         p /= pfac;
02735         t -= lapse_rate(t, h2o) * dz;
02736         psat = PSAT(t);
02737         h2o = psat / (p - (1. - EPS) * psat);
02738         INTPOL_INIT;
02739         intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
02740                             &t_env, ci, cw, 1);
02741         intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
02742                             &h2o_env, ci, cw, 0);
02743         dcape_old = dcape;
02744         dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
02745             TVIRT(t_env, h2o_env) * dz;
02746         if (dcape > 0) {
02747             met->cape[ix][iy] += (float) dcape;
02748             if (!isfinite(met->plfc[ix][iy]))
02749                 met->plfc[ix][iy] = (float) p;
02750             } else if (dcape_old > 0)
02751                 met->pel[ix][iy] = (float) p;
02752         } while (p > ptop);
02753     }
02754 }

```

Here is the call graph for this function:



**5.21.2.24 read\_met\_cloud()** void read\_met\_cloud (  
     met\_t \* met )

Calculate cloud properties.

Definition at line 2758 of file libtrac.c.

```

02759     {
02760
02761     /* Set timer... */
02762     SELECT_TIMER("READ_MET_CLOUD", NVTX_READ);
02763
02764     /* Loop over columns... */
02765     #pragma omp parallel for default(shared)
02766     for (int ix = 0; ix < met->nx; ix++)
02767         for (int iy = 0; iy < met->ny; iy++) {
02768
02769         /* Init... */
02770         met->pc[ix][iy] = GSL_NAN;
02771         met->cl[ix][iy] = 0;
02772
02773         /* Loop over pressure levels... */
02774         for (int ip = 0; ip < met->np - 1; ip++) {
02775
02776         /* Check pressure... */
02777         if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
02778             continue;
02779
02780         /* Get cloud top pressure ... */
02781         if (met->iwc[ix][iy][ip] > 0 || met->lwc[ix][iy][ip] > 0)
02782             met->pc[ix][iy] = (float) met->p[ip + 1];
02783
02784         /* Get cloud water... */
02785         met->cl[ix][iy] += (float)
02786             (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1]
02787                 + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
02788              * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
02789         }
02790     }
02791 }
```

**5.21.2.25 read\_met\_detrend()** void read\_met\_detrend (  
     ctl\_t \* ctl,  
     met\_t \* met )

Apply detrending method to temperature and winds.

Definition at line 2795 of file libtrac.c.

```

02797     {
02798
02799     met_t *help;
02800
02801     /* Check parameters... */
02802     if (ctl->met_detrend <= 0)
02803         return;
02804
02805     /* Set timer... */
02806     SELECT_TIMER("READ_MET_DETREND", NVTX_READ);
02807
02808     /* Allocate... */
02809     ALLOC(help, met_t, 1);
02810
02811     /* Calculate standard deviation... */
02812     double sigma = ctl->met_detrend / 2.355;
02813     double tssq = 2. * SQR(sigma);
02814
02815     /* Calculate box size in latitude... */
02816     int sy = (int) (3. * DY2DEG(sigma) / fabs(met->lat[1] - met->lat[0]));
02817     sy = GSL_MIN(GSL_MAX(1, sy), met->ny / 2);
02818
02819     /* Calculate background... */
02820     #pragma omp parallel for default(shared)
02821     for (int ix = 0; ix < met->nx; ix++) {
02822         for (int iy = 0; iy < met->ny; iy++) {
02823
02824         /* Calculate Cartesian coordinates... */
02825         double x0[3];
```

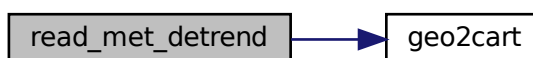


```

02826     geo2cart(0.0, met->lon[ix], met->lat[iy], x0);
02827
02828     /* Calculate box size in longitude... */
02829     int sx =
02830         (int) (3. * DX2DEG(sigma, met->lat[iy]) /
02831             fabs(met->lon[1] - met->lon[0]));
02832     sx = GSL_MIN(GSL_MAX(1, sx), met->nx / 2);
02833
02834     /* Init... */
02835     float wsum = 0;
02836     for (int ip = 0; ip < met->np; ip++) {
02837         help->t[ix][iy][ip] = 0;
02838         help->u[ix][iy][ip] = 0;
02839         help->v[ix][iy][ip] = 0;
02840         help->w[ix][iy][ip] = 0;
02841     }
02842
02843     /* Loop over neighboring grid points... */
02844     for (int ix2 = ix - sx; ix2 <= ix + sx; ix2++) {
02845         int ix3 = ix2;
02846         if (ix3 < 0)
02847             ix3 += met->nx;
02848         else if (ix3 >= met->nx)
02849             ix3 -= met->nx;
02850         for (int iy2 = GSL_MAX(iy - sy, 0);
02851             iy2 <= GSL_MIN(iy + sy, met->ny - 1); iy2++) {
02852
02853             /* Calculate Cartesian coordinates... */
02854             double xl[3];
02855             geo2cart(0.0, met->lon[ix3], met->lat[iy2], xl);
02856
02857             /* Calculate weighting factor... */
02858             float w = (float) exp(-DIST2(x0, xl) / tssq);
02859
02860             /* Add data... */
02861             wsum += w;
02862             for (int ip = 0; ip < met->np; ip++) {
02863                 help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip];
02864                 help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip];
02865                 help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip];
02866                 help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip];
02867             }
02868         }
02869     }
02870
02871     /* Normalize... */
02872     for (int ip = 0; ip < met->np; ip++) {
02873         help->t[ix][iy][ip] /= wsum;
02874         help->u[ix][iy][ip] /= wsum;
02875         help->v[ix][iy][ip] /= wsum;
02876         help->w[ix][iy][ip] /= wsum;
02877     }
02878 }
02879 }
02880
02881 /* Subtract background... */
02882 #pragma omp parallel for default(shared)
02883 for (int ix = 0; ix < met->nx; ix++)
02884     for (int iy = 0; iy < met->ny; iy++)
02885         for (int ip = 0; ip < met->np; ip++) {
02886             met->t[ix][iy][ip] -= help->t[ix][iy][ip];
02887             met->u[ix][iy][ip] -= help->u[ix][iy][ip];
02888             met->v[ix][iy][ip] -= help->v[ix][iy][ip];
02889             met->w[ix][iy][ip] -= help->w[ix][iy][ip];
02890         }
02891
02892 /* Free... */
02893 free(help);
02894 }

```

Here is the call graph for this function:



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

Extrapolate meteorological data at lower boundary.

Definition at line 2898 of file libtrac.c.

```

02899     {
02900
02901     int ip, ip0, ix, iy;
02902
02903     /* Set timer... */
02904     SELECT_TIMER("READ_MET_EXTRAPOLATE", NVTX_READ);
02905
02906     /* Loop over columns... */
02907     #pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
02908     for (ix = 0; ix < met->nx; ix++)
02909         for (iy = 0; iy < met->ny; iy++) {
02910
02911         /* Find lowest valid data point... */
02912         for (ip0 = met->np - 1; ip0 >= 0; ip0--)
02913             if (!isfinite(met->t[ix][iy][ip0])
02914                 || !isfinite(met->u[ix][iy][ip0])
02915                 || !isfinite(met->v[ix][iy][ip0])
02916                 || !isfinite(met->w[ix][iy][ip0]))
02917                 break;
02918
02919         /* Extrapolate... */
02920         for (ip = ip0; ip >= 0; ip--) {
02921             met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
02922             met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
02923             met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
02924             met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
02925             met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
02926             met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
02927             met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
02928             met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
02929         }
02930     }
02931 }
```

**5.21.2.27 read\_met\_geopot()** void read\_met\_geopot (  
     ctl\_t \* ctl,  
     met\_t \* met )

Calculate geopotential heights.

Definition at line 2935 of file libtrac.c.

```

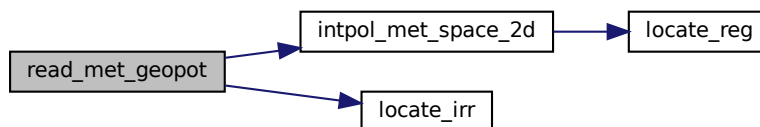
02937     {
02938
02939     const int dx = ctl->met_geopot_sx, dy = ctl->met_geopot_sy;
02940
02941     static float help[EX][EY][EP], w, wsum;
02942
02943     double h2os, logp[EP], ts, z0;
02944
02945     int ip, ip0, ix, ix2, ix3, iy, iy2;
02946
02947     /* Set timer... */
02948     SELECT_TIMER("READ_MET_GEOPOT", NVTX_READ);
02949
02950     /* Calculate log pressure... */
02951     for (ip = 0; ip < met->np; ip++)
02952         logp[ip] = log(met->p[ip]);
02953
02954     /* Initialize geopotential heights... */
02955     #pragma omp parallel for default(shared) private(ix,iy,ip)
02956     for (ix = 0; ix < met->nx; ix++)
02957         for (iy = 0; iy < met->ny; iy++)
02958             for (ip = 0; ip < met->np; ip++)
02959                 met->z[ix][iy][ip] = GSL_NAN;
02960 }
```

```

02961  /* Apply hydrostatic equation to calculate geopotential heights... */
02962 #pragma omp parallel for default(shared) private(ix,iy,z0,ip0,ts,ip)
02963 for (ix = 0; ix < met->nx; ix++)
02964     for (iy = 0; iy < met->ny; iy++) {
02965
02966         /* Get surface height... */
02967         INTPOL_INIT;
02968         intpol_met_space_2d(met, met->z0, met->lon[ix], met->lat[iy], &z0, ci,
02969                             cw, 1);
02970
02971         /* Find surface pressure level index... */
02972         ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
02973
02974         /* Get temperature and water vapor vmr at the surface... */
02975         ts =
02976             LIN(met->p[ip0], met->t[ix][iy][ip0], met->p[ip0 + 1],
02977                 met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
02978         h2os =
02979             LIN(met->p[ip0], met->h2o[ix][iy][ip0], met->p[ip0 + 1],
02980                 met->h2o[ix][iy][ip0 + 1], met->ps[ix][iy]);
02981
02982         /* Upper part of profile... */
02983         met->z[ix][iy][ip0 + 1]
02984             = (float) (z0 +
02985                     ZDIFF(log(met->ps[ix][iy]), ts, h2os, logp[ip0 + 1],
02986                           met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]));
02987         for (ip = ip0 + 2; ip < met->np; ip++)
02988             met->z[ix][iy][ip]
02989                 = (float) (met->z[ix][iy][ip - 1] +
02990                           ZDIFF(logp[ip - 1], met->t[ix][iy][ip - 1],
02991                                 met->h2o[ix][iy][ip - 1], logp[ip],
02992                                 met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
02993
02994         /* Lower part of profile... */
02995         met->z[ix][iy][ip0]
02996             = (float) (z0 +
02997                     ZDIFF(log(met->ps[ix][iy]), ts, h2os, logp[ip0],
02998                           met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]));
02999         for (ip = ip0 - 1; ip >= 0; ip--)
03000             met->z[ix][iy][ip]
03001                 = (float) (met->z[ix][iy][ip + 1] +
03002                           ZDIFF(logp[ip + 1], met->t[ix][iy][ip + 1],
03003                                 met->h2o[ix][iy][ip + 1], logp[ip],
03004                                 met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
03005     }
03006
03007     /* Horizontal smoothing... */
03008 #pragma omp parallel for default(shared) private(ix,iy,ip,ix2,ix3,iy2,w,wsum)
03009     for (ix = 0; ix < met->nx; ix++)
03010         for (iy = 0; iy < met->ny; iy++)
03011             for (ip = 0; ip < met->np; ip++) {
03012                 wsum = 0;
03013                 help[ix][iy][ip] = 0;
03014                 for (ix2 = ix - dx + 1; ix2 <= ix + dx - 1; ix2++) {
03015                     ix3 = ix2;
03016                     if (ix3 < 0)
03017                         ix3 += met->nx;
03018                     else if (ix3 >= met->nx)
03019                         ix3 -= met->nx;
03020                     for (iy2 = GSL_MAX(iy - dy + 1, 0);
03021                         iy2 <= GSL_MIN(iy + dy - 1, met->ny - 1); iy2++)
03022                         if (isfinite(met->z[ix3][iy2][ip])) {
03023                             w = (1.0f - (float) abs(ix - ix2) / (float) dx)
03024                                 * (1.0f - (float) abs(iy - iy2) / (float) dy);
03025                             help[ix][iy][ip] += w * met->z[ix3][iy2][ip];
03026                             wsum += w;
03027                         }
03028                 }
03029                 if (wsum > 0)
03030                     help[ix][iy][ip] /= wsum;
03031                 else
03032                     help[ix][iy][ip] = GSL_NAN;
03033             }
03034
03035     /* Copy data... */
03036 #pragma omp parallel for default(shared) private(ix,iy,ip)
03037     for (ix = 0; ix < met->nx; ix++)
03038         for (iy = 0; iy < met->ny; iy++)
03039             for (ip = 0; ip < met->np; ip++)
03040                 met->z[ix][iy][ip] = help[ix][iy][ip];
03041 }

```

Here is the call graph for this function:



**5.21.2.28 read\_met\_grid()** void read\_met\_grid (

```

    char * filename,
    int ncid,
    ctl_t * ctl,
    met_t * met )

```

Read coordinates of meteorological data.

Definition at line 3045 of file libtrac.c.

```

03049     {
03050
03051     char levname[LEN], tstr[10];
03052
03053     int dimid, ip, varid, year, mon, day, hour;
03054
03055     size_t np, nx, ny;
03056
03057     /* Set timer... */
03058     SELECT_TIMER("READ_MET_GRID", NVTX_READ);
03059
03060     /* Get time from filename... */
03061     sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
03062     year = atoi(tstr);
03063     sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
03064     mon = atoi(tstr);
03065     sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
03066     day = atoi(tstr);
03067     sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
03068     hour = atoi(tstr);
03069     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
03070
03071     /* Get grid dimensions... */
03072     NC(nc_inq_dimid(ncid, "lon", &dimid));
03073     NC(nc_inq_dimlen(ncid, dimid, &nx));
03074     if (nx < 2 || nx > EX)
03075         ERRMSG("Number of longitudes out of range!");
03076
03077     NC(nc_inq_dimid(ncid, "lat", &dimid));
03078     NC(nc_inq_dimlen(ncid, dimid, &ny));
03079     if (ny < 2 || ny > EY)
03080         ERRMSG("Number of latitudes out of range!");
03081
03082     sprintf(levname, "lev");
03083     NC(nc_inq_dimid(ncid, levname, &dimid));
03084     NC(nc_inq_dimlen(ncid, dimid, &np));
03085     if (np == 1) {
03086         sprintf(levname, "lev_2");
03087         if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
03088             sprintf(levname, "plev");
03089             nc_inq_dimid(ncid, levname, &dimid);
03090         }
03091         NC(nc_inq_dimlen(ncid, dimid, &np));
03092     }
03093     if (np < 2 || np > EP)
03094         ERRMSG("Number of levels out of range!");
03095
03096     /* Store dimensions... */

```

```

03097 met->np = (int) np;
03098 met->nx = (int) nx;
03099 met->ny = (int) ny;
03100
03101 /* Read longitudes and latitudes... */
03102 NC(nc_inq_varid(ncid, "lon", &varid));
03103 NC(nc_get_var_double(ncid, varid, met->lon));
03104 NC(nc_inq_varid(ncid, "lat", &varid));
03105 NC(nc_get_var_double(ncid, varid, met->lat));
03106
03107 /* Read pressure levels... */
03108 if (ctl->met_np <= 0) {
03109     NC(nc_inq_varid(ncid, levname, &varid));
03110     NC(nc_get_var_double(ncid, varid, met->p));
03111     for (ip = 0; ip < met->np; ip++)
03112         met->p[ip] /= 100.;
03113 }
03114
03115 /* Set pressure levels... */
03116 else {
03117     met->np = ctl->met_np;
03118     for (ip = 0; ip < met->np; ip++)
03119         met->p[ip] = ctl->met_p[ip];
03120 }
03121
03122 /* Check ordering of pressure levels... */
03123 for (ip = 1; ip < met->np; ip++)
03124     if (met->p[ip - 1] < met->p[ip])
03125         ERRMSG("Pressure levels must be descending!");
03126 }

```

Here is the call graph for this function:



**5.21.2.29 read\_met\_help\_3d()** int read\_met\_help\_3d (

```

    int ncid,
    char * varname,
    char * varname2,
    met_t * met,
    float dest[EX][EY][EP],
    float scl )

```

Read and convert 3D variable from meteorological data file.

Definition at line 3130 of file [libtrac.c](#).

```

03136 {
03137
03138     float *help;
03139
03140     int ip, ix, iy, varid;
03141
03142     /* Check if variable exists... */
03143     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03144         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
03145             return 0;
03146
03147     /* Allocate... */
03148     ALLOC(help, float,
03149           EX * EY * EP);
03150

```

```

03151  /* Read data... */
03152  NC(nc_get_var_float(ncid, varid, help));
03153
03154  /* Copy and check data... */
03155  #pragma omp parallel for default(shared) private(ix,iy,ip)
03156  for (ix = 0; ix < met->nx; ix++)
03157      for (iy = 0; iy < met->ny; iy++)
03158          for (ip = 0; ip < met->np; ip++) {
03159              dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
03160              if (fabsf(dest[ix][iy][ip]) < 1e14f)
03161                  dest[ix][iy][ip] *= scl;
03162              else
03163                  dest[ix][iy][ip] = GSL_NAN;
03164          }
03165
03166  /* Free... */
03167  free(help);
03168
03169  /* Return... */
03170  return 1;
03171 }

```

**5.21.2.30 read\_met\_help\_2d()** int read\_met\_help\_2d (

```

    int ncid,
    char * varname,
    char * varname2,
    met_t * met,
    float dest[EX][EY],
    float scl )

```

Read and convert 2D variable from meteorological data file.

Definition at line 3175 of file libtrac.c.

```

03181  {
03182
03183  float *help;
03184
03185  int ix, iy, varid;
03186
03187  /* Check if variable exists... */
03188  if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03189      if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
03190          return 0;
03191
03192  /* Allocate... */
03193  ALLOC(help, float,
03194        EX * EY);
03195
03196  /* Read data... */
03197  NC(nc_get_var_float(ncid, varid, help));
03198
03199  /* Copy and check data... */
03200  #pragma omp parallel for default(shared) private(ix,iy)
03201  for (ix = 0; ix < met->nx; ix++)
03202      for (iy = 0; iy < met->ny; iy++) {
03203          dest[ix][iy] = help[iy * met->nx + ix];
03204          if (fabsf(dest[ix][iy]) < 1e14f)
03205              dest[ix][iy] *= scl;
03206          else
03207              dest[ix][iy] = GSL_NAN;
03208      }
03209
03210  /* Free... */
03211  free(help);
03212
03213  /* Return... */
03214  return 1;
03215 }

```

**5.21.2.31 read\_met\_levels()** void read\_met\_levels (

```

    int ncid,
    ctl_t * ctl,
    met_t * met )

```

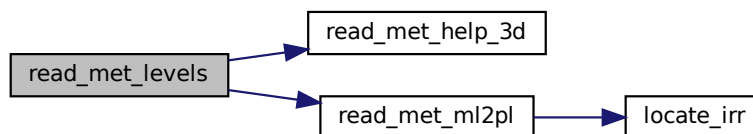
Definition at line 3219 of file libtrac.c.

```

03222     {
03223
03224     /* Set timer... */
03225     SELECT_TIMER("READ_MET_LEVELS", NVTX_READ);
03226
03227     /* Read meteorological data... */
03228     if (!read_met_help_3d(ncid, "t", "T", met, met->t, 1.0))
03229         ERRMSG("Cannot read temperature!");
03230     if (!read_met_help_3d(ncid, "u", "U", met, met->u, 1.0))
03231         ERRMSG("Cannot read zonal wind!");
03232     if (!read_met_help_3d(ncid, "v", "V", met, met->v, 1.0))
03233         ERRMSG("Cannot read meridional wind!");
03234     if (!read_met_help_3d(ncid, "w", "W", met, met->w, 0.01f))
03235         WARN("Cannot read vertical velocity");
03236     if (!read_met_help_3d(ncid, "q", "Q", met, met->h2o, (float) (MA / MH2O)))
03237         WARN("Cannot read specific humidity!");
03238     if (!read_met_help_3d(ncid, "o3", "O3", met, met->o3, (float) (MA / MO3)))
03239         WARN("Cannot read ozone data!");
03240     if (!read_met_help_3d(ncid, "clwc", "CLWC", met, met->lwc, 1.0))
03241         WARN("Cannot read cloud liquid water content!");
03242     if (!read_met_help_3d(ncid, "ciwc", "CIWC", met, met->iwc, 1.0))
03243         WARN("Cannot read cloud ice water content!");
03244
03245     /* Transfer from model levels to pressure levels... */
03246     if (ctl->met_np > 0) {
03247
03248         /* Read pressure on model levels... */
03249         if (!read_met_help_3d(ncid, "pl", "PL", met, met->pl, 0.01f))
03250             ERRMSG("Cannot read pressure on model levels!");
03251
03252         /* Vertical interpolation from model to pressure levels... */
03253         read_met_ml2pl(ctl, met, met->t);
03254         read_met_ml2pl(ctl, met, met->u);
03255         read_met_ml2pl(ctl, met, met->v);
03256         read_met_ml2pl(ctl, met, met->w);
03257         read_met_ml2pl(ctl, met, met->h2o);
03258         read_met_ml2pl(ctl, met, met->o3);
03259         read_met_ml2pl(ctl, met, met->lwc);
03260         read_met_ml2pl(ctl, met, met->iwc);
03261     }
03262 }

```

Here is the call graph for this function:



**5.21.2.32 read\_met\_ml2pl()** void read\_met\_ml2pl (

```

    ctl_t * ctl,
    met_t * met,
    float var[EX][EY][EP] )

```

Convert meteorological data from model levels to pressure levels.

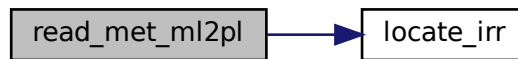
Definition at line 3266 of file libtrac.c.

```

03269     {
03270
03271     double aux[EP], p[EP], pt;
03272
03273     int ip, ip2, ix, iy;
03274
03275     /* Set timer... */
03276     SELECT_TIMER("READ_MET_ML2PL", NVTX_READ);
03277
03278     /* Loop over columns... */
03279     #pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
03280     for (ix = 0; ix < met->nx; ix++)
03281         for (iy = 0; iy < met->ny; iy++) {
03282
03283             /* Copy pressure profile... */
03284             for (ip = 0; ip < met->np; ip++)
03285                 p[ip] = met->p[ix][iy][ip];
03286
03287             /* Interpolate... */
03288             for (ip = 0; ip < ctl->met_np; ip++) {
03289                 pt = ctl->met_p[ip];
03290                 if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
03291                     pt = p[0];
03292                 else if ((pt > p[met->np - 1] && p[1] > p[0])
03293                        || (pt < p[met->np - 1] && p[1] < p[0]))
03294                     pt = p[met->np - 1];
03295                 ip2 = locate_irr(p, met->np, pt);
03296                 aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
03297                             p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
03298             }
03299
03300             /* Copy data... */
03301             for (ip = 0; ip < ctl->met_np; ip++)
03302                 var[ix][iy][ip] = (float) aux[ip];
03303         }
03304     }

```

Here is the call graph for this function:



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

Create meteorological data with periodic boundary conditions.

Definition at line 3308 of file libtrac.c.

```

03309     {
03310
03311     /* Set timer... */
03312     SELECT_TIMER("READ_MET_PERIODIC", NVTX_READ);
03313
03314     /* Check longitudes... */
03315     if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
03316              + met->lon[1] - met->lon[0] - 360) < 0.01))
03317         return;
03318
03319     /* Increase longitude counter... */
03320     if ((++met->nx) > EX)
03321         ERRMSG("Cannot create periodic boundary conditions!");
03322
03323     /* Set longitude... */

```



```

03324   met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->lon[0];
03325
03326   /* Loop over latitudes and pressure levels... */
03327 #pragma omp parallel for default(shared)
03328   for (int iy = 0; iy < met->ny; iy++) {
03329       met->ps[met->nx - 1][iy] = met->ps[0][iy];
03330       met->zs[met->nx - 1][iy] = met->zs[0][iy];
03331       for (int ip = 0; ip < met->np; ip++) {
03332           met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
03333           met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
03334           met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
03335           met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
03336           met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
03337           met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
03338           met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
03339           met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
03340       }
03341   }
03342 }

```

**5.21.2.34 read\_met\_pv()** void read\_met\_pv (  
     met\_t \* met )

Calculate potential vorticity.

Definition at line 3346 of file libtrac.c.

```

03347   {
03348
03349       double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
03350           dtdp, dudp, dvdp, latr, vort, pows[EP];
03351
03352       int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
03353
03354       /* Set timer... */
03355       SELECT_TIMER("READ_MET_PV", NVTX_READ);
03356
03357       /* Set powers... */
03358       for (ip = 0; ip < met->np; ip++)
03359           pows[ip] = pow(1000. / met->p[ip], 0.286);
03360
03361       /* Loop over grid points... */
03362 #pragma omp parallel for default(shared)
03363       private(ix,ix0,ix1,iy,iy0,iy1,latr,dx,dy,c0,c1,cr,vort,ip,ip0,ip1,dp0,dp1,denom,dtdx,dvdx,dtdy,dudy,dtdp,dudp,dvdp)
03364       for (ix = 0; ix < met->nx; ix++) {
03365
03366           /* Set indices... */
03367           ix0 = GSL_MAX(ix - 1, 0);
03368           ix1 = GSL_MIN(ix + 1, met->nx - 1);
03369
03370           /* Loop over grid points... */
03371           for (iy = 0; iy < met->ny; iy++) {
03372
03373               /* Set indices... */
03374               iy0 = GSL_MAX(iy - 1, 0);
03375               iy1 = GSL_MIN(iy + 1, met->ny - 1);
03376
03377               /* Set auxiliary variables... */
03378               latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
03379               dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
03380               dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
03381               c0 = cos(met->lat[iy0] / 180. * M_PI);
03382               c1 = cos(met->lat[iy1] / 180. * M_PI);
03383               cr = cos(latr / 180. * M_PI);
03384               vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
03385
03386               /* Loop over grid points... */
03387               for (ip = 0; ip < met->np; ip++) {
03388
03389                   /* Get gradients in longitude... */
03390                   dtdx = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
03391                   dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
03392
03393                   /* Get gradients in latitude... */
03394                   dtdy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
03395                   dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
03396
03397                   /* Set indices... */
03398                   ip0 = GSL_MAX(ip - 1, 0);
03399                   ip1 = GSL_MIN(ip + 1, met->np - 1);

```

```

03399
03400      /* Get gradients in pressure... */
03401      dp0 = 100. * (met->p[ip] - met->p[ip0]);
03402      dp1 = 100. * (met->p[ip1] - met->p[ip]);
03403      if (ip != ip0 && ip != ip1) {
03404          denom = dp0 * dp1 * (dp0 + dp1);
03405          dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
03406                - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
03407                + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
03408                / denom;
03409          dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
03410                - dp1 * dp1 * met->u[ix][iy][ip0]
03411                + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
03412                / denom;
03413          dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
03414                - dp1 * dp1 * met->v[ix][iy][ip0]
03415                + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
03416                / denom;
03417      } else {
03418          denom = dp0 + dp1;
03419          dtdp =
03420              (met->t[ix][iy][ip1] * pows[ip1] -
03421               met->t[ix][iy][ip0] * pows[ip0]) / denom;
03422          dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
03423          dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
03424      }
03425
03426      /* Calculate PV... */
03427      met->pv[ix][iy][ip] = (float)
03428          (1e6 * G0 *
03429           (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
03430  }
03431 }
03432 }
03433
03434 /* Fix for polar regions... */
03435 #pragma omp parallel for default(shared) private(ix,ip)
03436 for (ix = 0; ix < met->nx; ix++)
03437     for (ip = 0; ip < met->np; ip++) {
03438         met->pv[ix][0][ip]
03439             = met->pv[ix][1][ip]
03440             = met->pv[ix][2][ip];
03441         met->pv[ix][met->ny - 1][ip]
03442             = met->pv[ix][met->ny - 2][ip]
03443             = met->pv[ix][met->ny - 3][ip];
03444     }
03445 }

```

**5.21.2.35 read\_met\_sample()** void read\_met\_sample (  
     ctl\_t \* ctl,  
     met\_t \* met )

Downsampling of meteorological data.

Definition at line 3449 of file libtrac.c.

```

03451     {
03452
03453         met_t *help;
03454
03455         float w, wsum;
03456
03457         int ip, ip2, ix, ix2, ix3, iy, iy2;
03458
03459         /* Check parameters... */
03460         if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
03461             && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
03462             return;
03463
03464         /* Set timer... */
03465         SELECT_TIMER("READ_MET_SAMPLE", NVTX_READ);
03466
03467         /* Allocate... */
03468         ALLOC(help, met_t, 1);
03469
03470         /* Copy data... */
03471         help->nx = met->nx;
03472         help->ny = met->ny;
03473         help->np = met->np;

```

```

03474 memcpy(help->lon, met->lon, sizeof(met->lon));
03475 memcpy(help->lat, met->lat, sizeof(met->lat));
03476 memcpy(help->p, met->p, sizeof(met->p));
03477
03478 /* Smoothing... */
03479 for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
03480     for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
03481         for (ip = 0; ip < met->np; ip += ctl->met_dp) {
03482             help->ps[ix][iy] = 0;
03483             help->zs[ix][iy] = 0;
03484             help->t[ix][iy][ip] = 0;
03485             help->u[ix][iy][ip] = 0;
03486             help->v[ix][iy][ip] = 0;
03487             help->w[ix][iy][ip] = 0;
03488             help->h2o[ix][iy][ip] = 0;
03489             help->o3[ix][iy][ip] = 0;
03490             help->lwc[ix][iy][ip] = 0;
03491             help->iwc[ix][iy][ip] = 0;
03492             wsum = 0;
03493             for (ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1; ix2++) {
03494                 ix3 = ix2;
03495                 if (ix3 < 0)
03496                     ix3 += met->nx;
03497                 else if (ix3 >= met->nx)
03498                     ix3 -= met->nx;
03499
03500                 for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
03501                     iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
03502                     for (ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
03503                         ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
03504                         w = (1.0f - (float) abs(ix - ix2) / (float) ctl->met_sx)
03505                             * (1.0f - (float) abs(iy - iy2) / (float) ctl->met_sy)
03506                             * (1.0f - (float) abs(ip - ip2) / (float) ctl->met_sp);
03507                         help->ps[ix][iy] += w * met->ps[ix3][iy2];
03508                         help->zs[ix][iy] += w * met->zs[ix3][iy2];
03509                         help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
03510                         help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
03511                         help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
03512                         help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
03513                         help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
03514                         help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
03515                         help->lwc[ix][iy][ip] += w * met->lwc[ix3][iy2][ip2];
03516                         help->iwc[ix][iy][ip] += w * met->iwc[ix3][iy2][ip2];
03517                         wsum += w;
03518                     }
03519             }
03520             help->ps[ix][iy] /= wsum;
03521             help->zs[ix][iy] /= wsum;
03522             help->t[ix][iy][ip] /= wsum;
03523             help->u[ix][iy][ip] /= wsum;
03524             help->v[ix][iy][ip] /= wsum;
03525             help->w[ix][iy][ip] /= wsum;
03526             help->h2o[ix][iy][ip] /= wsum;
03527             help->o3[ix][iy][ip] /= wsum;
03528             help->lwc[ix][iy][ip] /= wsum;
03529             help->iwc[ix][iy][ip] /= wsum;
03530         }
03531     }
03532 }
03533
03534 /* Downsampling... */
03535 met->nx = 0;
03536 for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
03537     met->lon[met->nx] = help->lon[ix];
03538     met->ny = 0;
03539     for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
03540         met->lat[met->ny] = help->lat[iy];
03541         met->ps[met->nx][met->ny] = help->ps[ix][iy];
03542         met->zs[met->nx][met->ny] = help->zs[ix][iy];
03543         met->np = 0;
03544         for (ip = 0; ip < help->np; ip += ctl->met_dp) {
03545             met->p[met->np] = help->p[ip];
03546             met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
03547             met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
03548             met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
03549             met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
03550             met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
03551             met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
03552             met->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];
03553             met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
03554             met->np++;
03555         }
03556         met->ny++;
03557     }
03558     met->nx++;
03559 }
03560

```

```

03561  /* Free... */
03562  free(help);
03563  }

```

**5.21.2.36 read\_met\_surface()** void read\_met\_surface (

```

    int ncid,
    met_t * met )

```

Read surface data.

Definition at line 3567 of file libtrac.c.

```

03569  {
03570
03571  int ix, iy;
03572
03573  /* Set timer... */
03574  SELECT_TIMER("READ_MET_SURFACE", NVTX_READ);
03575
03576  /* Read surface pressure... */
03577  if (!read_met_help_2d(ncid, "ps", "PS", met, met->ps, 0.01f)) {
03578      if (!read_met_help_2d(ncid, "lnsp", "LNSP", met, met->ps, 1.0)) {
03579          ERRMSG("Cannot not read surface pressure data!");
03580          for (ix = 0; ix < met->nx; ix++)
03581              for (iy = 0; iy < met->ny; iy++)
03582                  met->ps[ix][iy] = (float) met->p[0];
03583      } else {
03584          for (iy = 0; iy < met->ny; iy++)
03585              for (ix = 0; ix < met->nx; ix++)
03586                  met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
03587      }
03588  }
03589
03590  /* Read geopotential height at the surface... */
03591  if (!read_met_help_2d
03592      (ncid, "z", "Z", met, met->zs, (float) (1. / (1000. * G0))))
03593      if (!read_met_help_2d
03594          (ncid, "zm", "ZM", met, met->zs, (float) (1. / 1000.)))
03595          ERRMSG("Cannot read surface geopotential height!");
03596
03597  /* Read temperature at the surface... */
03598  if (!read_met_help_2d(ncid, "t2m", "T2M", met, met->ts, 1.0))
03599      WARN("Cannot read surface temperature!");
03600
03601  /* Read zonal wind at the surface... */
03602  if (!read_met_help_2d(ncid, "u10m", "U10M", met, met->us, 1.0))
03603      WARN("Cannot read surface zonal wind!");
03604
03605  /* Read meridional wind at the surface... */
03606  if (!read_met_help_2d(ncid, "v10m", "V10M", met, met->vs, 1.0))
03607      WARN("Cannot read surface meridional wind!");
03608  }

```

Here is the call graph for this function:



**5.21.2.37 read\_met\_tropo()** void read\_met\_tropo (  
     ctl\_t \* ctl,  
     met\_t \* met )

Calculate tropopause data.

Definition at line 3612 of file libtrac.c.

```

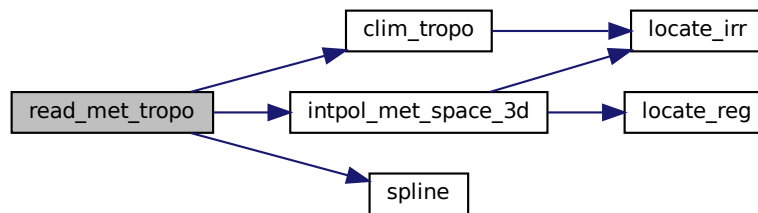
03614     {
03615
03616     double h2ot, p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
03617         th2[200], tt, z[EP], z2[200], zt;
03618
03619     int found, ix, iy, iz, iz2;
03620
03621     /* Set timer... */
03622     SELECT_TIMER("READ_MET_TROPO", NVTX_READ);
03623
03624     /* Get altitude and pressure profiles... */
03625     for (iz = 0; iz < met->np; iz++)
03626         z[iz] = Z(met->p[iz]);
03627     for (iz = 0; iz <= 190; iz++) {
03628         z2[iz] = 4.5 + 0.1 * iz;
03629         p2[iz] = P(z2[iz]);
03630     }
03631
03632     /* Do not calculate tropopause... */
03633     if (ctl->met_tropo == 0)
03634         for (ix = 0; ix < met->nx; ix++)
03635             for (iy = 0; iy < met->ny; iy++)
03636                 met->pt[ix][iy] = GSL_NAN;
03637
03638     /* Use tropopause climatology... */
03639     else if (ctl->met_tropo == 1) {
03640 #pragma omp parallel for default(shared) private(ix,iy)
03641         for (ix = 0; ix < met->nx; ix++)
03642             for (iy = 0; iy < met->ny; iy++)
03643                 met->pt[ix][iy] = (float) clim_tropo(met->time, met->lat[iy]);
03644     }
03645
03646     /* Use cold point... */
03647     else if (ctl->met_tropo == 2) {
03648
03649         /* Loop over grid points... */
03650 #pragma omp parallel for default(shared) private(ix,iy,iz,t,t2)
03651         for (ix = 0; ix < met->nx; ix++)
03652             for (iy = 0; iy < met->ny; iy++) {
03653
03654                 /* Interpolate temperature profile... */
03655                 for (iz = 0; iz < met->np; iz++)
03656                     t[iz] = met->t[ix][iy][iz];
03657                 spline(z, t, met->np, z2, t2, 171);
03658
03659                 /* Find minimum... */
03660                 iz = (int) gsl_stats_min_index(t2, 1, 171);
03661                 if (iz > 0 && iz < 170)
03662                     met->pt[ix][iy] = (float) p2[iz];
03663                 else
03664                     met->pt[ix][iy] = GSL_NAN;
03665             }
03666     }
03667
03668     /* Use WMO definition... */
03669     else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
03670
03671         /* Loop over grid points... */
03672 #pragma omp parallel for default(shared) private(ix,iy,iz,iz2,t,t2,found)
03673         for (ix = 0; ix < met->nx; ix++)
03674             for (iy = 0; iy < met->ny; iy++) {
03675
03676                 /* Interpolate temperature profile... */
03677                 for (iz = 0; iz < met->np; iz++)
03678                     t[iz] = met->t[ix][iy][iz];
03679                 spline(z, t, met->np, z2, t2, 191);
03680
03681                 /* Find 1st tropopause... */
03682                 met->pt[ix][iy] = GSL_NAN;
03683                 for (iz = 0; iz <= 170; iz++) {
03684                     found = 1;
03685                     for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
03686                         if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) > 2.0) {
03687                             found = 0;
03688                             break;
03689                         }
03690                     if (found) {
```

```

03691         if (iz > 0 && iz < 170)
03692             met->pt[ix][iy] = (float) p2[iz];
03693         break;
03694     }
03695 }
03696
03697 /* Find 2nd tropopause... */
03698 if (ctl->met_tropo == 4) {
03699     met->pt[ix][iy] = GSL_NAN;
03700     for (; iz <= 170; iz++) {
03701         found = 1;
03702         for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
03703             if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) < 3.0) {
03704                 found = 0;
03705                 break;
03706             }
03707         if (found)
03708             break;
03709     }
03710     for (; iz <= 170; iz++) {
03711         found = 1;
03712         for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
03713             if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) > 2.0) {
03714                 found = 0;
03715                 break;
03716             }
03717         if (found) {
03718             if (iz > 0 && iz < 170)
03719                 met->pt[ix][iy] = (float) p2[iz];
03720             break;
03721         }
03722     }
03723 }
03724 }
03725 }
03726
03727 /* Use dynamical tropopause... */
03728 else if (ctl->met_tropo == 5) {
03729     /* Loop over grid points... */
03730     #pragma omp parallel for default(shared) private(ix,iy,iz,pv,pv2,th,th2)
03731     for (ix = 0; ix < met->nx; ix++)
03732         for (iy = 0; iy < met->ny; iy++) {
03733             /* Interpolate potential vorticity profile... */
03734             for (iz = 0; iz < met->np; iz++)
03735                 pv[iz] = met->pv[ix][iy][iz];
03736             spline(z, pv, met->np, z2, pv2, 171);
03737
03738             /* Interpolate potential temperature profile... */
03739             for (iz = 0; iz < met->np; iz++)
03740                 th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
03741             spline(z, th, met->np, z2, th2, 171);
03742
03743             /* Find dynamical tropopause 3.5 PVU + 380 K */
03744             met->pt[ix][iy] = GSL_NAN;
03745             for (iz = 0; iz <= 170; iz++)
03746                 if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
03747                     if (iz > 0 && iz < 170)
03748                         met->pt[ix][iy] = (float) p2[iz];
03749                     break;
03750                 }
03751         }
03752     }
03753 }
03754 }
03755
03756 else
03757     ERRMSG("Cannot calculate tropopause!");
03758
03759 /* Interpolate temperature, geopotential height, and water vapor vmr... */
03760 #pragma omp parallel for default(shared) private(ix,iy,tt,zt,h2ot)
03761 for (ix = 0; ix < met->nx; ix++)
03762     for (iy = 0; iy < met->ny; iy++) {
03763         INTPOL_INIT;
03764         intpol_met_space_3d(met, met->t, met->pt[ix][iy], met->lon[ix],
03765                             met->lat[iy], &tt, ci, cw, 1);
03766         intpol_met_space_3d(met, met->z, met->pt[ix][iy], met->lon[ix],
03767                             met->lat[iy], &zt, ci, cw, 0);
03768         intpol_met_space_3d(met, met->h2o, met->pt[ix][iy], met->lon[ix],
03769                             met->lat[iy], &h2ot, ci, cw, 0);
03770         met->tt[ix][iy] = (float) tt;
03771         met->zt[ix][iy] = (float) zt;
03772         met->h2ot[ix][iy] = (float) h2ot;
03773     }
03774 }

```

Here is the call graph for this function:



**5.21.2.38 scan\_ctl()** 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 3778 of file libtrac.c.

```

03785     {
03786
03787     FILE *in = NULL;
03788
03789     char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
03790         msg[2 * LEN], rvarname[LEN], rval[LEN];
03791
03792     int contain = 0, i;
03793
03794     /* Open file... */
03795     if (filename[strlen(filename) - 1] != '-')
03796         if (!(in = fopen(filename, "r")))
03797             ERRMSG("Cannot open file!");
03798
03799     /* Set full variable name... */
03800     if (arridx >= 0) {
03801         sprintf(fullname1, "%s[%d]", varname, arridx);
03802         sprintf(fullname2, "%s[*]", varname);
03803     } else {
03804         sprintf(fullname1, "%s", varname);
03805         sprintf(fullname2, "%s", varname);
03806     }
03807
03808     /* Read data... */
03809     if (in != NULL)
03810         while (fgets(line, LEN, in))
03811             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
03812                 if (strcasecmp(rvarname, fullname1) == 0 ||
03813                     strcasecmp(rvarname, fullname2) == 0) {
03814                     contain = 1;
03815                     break;
03816                 }
03817     for (i = 1; i < argc - 1; i++)
03818         if (strcasecmp(argv[i], fullname1) == 0 ||
03819             strcasecmp(argv[i], fullname2) == 0) {
03820             sprintf(rval, "%s", argv[i + 1]);
03821             contain = 1;
03822             break;
03823         }
03824

```

```

03825  /* Close file... */
03826  if (in != NULL)
03827      fclose(in);
03828
03829  /* Check for missing variables... */
03830  if (!contain) {
03831      if (strlen(defvalue) > 0)
03832          sprintf(rval, "%s", defvalue);
03833      else {
03834          sprintf(msg, "Missing variable %s!\n", fullname1);
03835          ERRMSG(msg);
03836      }
03837  }
03838
03839  /* Write info... */
03840  printf("%s = %s\n", fullname1, rval);
03841
03842  /* Return values... */
03843  if (value != NULL)
03844      sprintf(value, "%s", rval);
03845  return atof(rval);
03846 }

```

**5.21.2.39 sedi()** double sedi (  
     double *p*,  
     double *T*,  
     double *r\_p*,  
     double *rho\_p* )

Calculate sedimentation velocity.

Definition at line 3850 of file libtrac.c.

```

03854  {
03855
03856  double eta, G, K, lambda, rho, v;
03857
03858  /* Convert units... */
03859  p *= 100.;                      /* from hPa to Pa */
03860  r_p *= 1e-6;                    /* from microns to m */
03861
03862  /* Density of dry air... */
03863  rho = p / (RA * T);
03864
03865  /* Dynamic viscosity of air... */
03866  eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
03867
03868  /* Thermal velocity of an air molecule... */
03869  v = sqrt(8. * KB * T / (M_PI * 4.8096e-26));
03870
03871  /* Mean free path of an air molecule... */
03872  lambda = 2. * eta / (rho * v);
03873
03874  /* Knudsen number for air... */
03875  K = lambda / r_p;
03876
03877  /* Cunningham slip-flow correction... */
03878  G = 1. + K * (1.249 + 0.42 * exp(-0.87 / K));
03879
03880  /* Sedimentation velocity... */
03881  return 2. * SQR(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;
03882 }

```

**5.21.2.40 spline()** void spline (  
     double \* *x*,  
     double \* *y*,  
     int *n*,  
     double \* *x2*,  
     double \* *y2*,  
     int *n2* )



Spline interpolation.

Definition at line 3886 of file [libtrac.c](#).

```
03892     {
03893
03894     gsl_interp_accel *acc;
03895
03896     gsl_spline *s;
03897
03898     /* Allocate... */
03899     acc = gsl_interp_accel_alloc();
03900     s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
03901
03902     /* Interpolate profile... */
03903     gsl_spline_init(s, x, y, (size_t) n);
03904     for (int i = 0; i < n2; i++)
03905         if (x2[i] <= x[0])
03906             y2[i] = y[0];
03907         else if (x2[i] >= x[n - 1])
03908             y2[i] = y[n - 1];
03909         else
03910             y2[i] = gsl_spline_eval(s, x2[i], acc);
03911
03912     /* Free... */
03913     gsl_spline_free(s);
03914     gsl_interp_accel_free(acc);
03915 }
```

**5.21.2.41 stddev()** double stddev (  
     double \* data,  
     int n )

Calculate standard deviation.

Definition at line 3919 of file [libtrac.c](#).

```
03921     {
03922
03923     if (n <= 0)
03924         return 0;
03925
03926     double avg = 0, rms = 0;
03927
03928     for (int i = 0; i < n; ++i)
03929         avg += data[i];
03930     avg /= n;
03931
03932     for (int i = 0; i < n; ++i)
03933         rms += SQR(data[i] - avg);
03934
03935     return sqrt(rms / (n - 1));
03936 }
```

**5.21.2.42 time2jsec()** 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 3940 of file [libtrac.c](#).

```
03948     {
```

```

03949
03950     struct tm t0, t1;
03951
03952     t0.tm_year = 100;
03953     t0.tm_mon = 0;
03954     t0.tm_mday = 1;
03955     t0.tm_hour = 0;
03956     t0.tm_min = 0;
03957     t0.tm_sec = 0;
03958
03959     t1.tm_year = year - 1900;
03960     t1.tm_mon = mon - 1;
03961     t1.tm_mday = day;
03962     t1.tm_hour = hour;
03963     t1.tm_min = min;
03964     t1.tm_sec = sec;
03965
03966     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
03967 }

```

**5.21.2.43 timer()** void timer (  
     const char \* name,  
     int output )

Measure wall-clock time.

Definition at line 3971 of file libtrac.c.

```

03973     {
03974
03975     static char namelist[NTIMER][100];
03976
03977     static double runtime[NTIMER], t0, t1;
03978
03979     static int it = -1, nt;
03980
03981     /* Get time... */
03982     t1 = omp_get_wtime();
03983
03984     /* Add elapsed timer to old timer... */
03985     if (it >= 0)
03986         runtime[it] += t1 - t0;
03987
03988     /* Identify ID of new timer... */
03989     for (it = 0; it < nt; it++)
03990         if (strcasemp(name, namelist[it]) == 0)
03991             break;
03992
03993     /* Check whether this is a new timer... */
03994     if (it >= nt) {
03995         sprintf(namelist[it], "%s", name);
03996         if ((++nt) > NTIMER)
03997             ERRMSG("Too many timers!");
03998     }
03999
04000     /* Save starting time... */
04001     t0 = t1;
04002
04003     /* Report timers... */
04004     if (output) {
04005         for (int it2 = 0; it2 < nt; it2++)
04006             printf("TIMER_%s = %.3f s\n", namelist[it2], runtime[it2]);
04007         double total = 0.0;
04008         for (int it2 = 0; it2 < nt; it2++)
04009             total += runtime[it2];
04010         printf("TIMER_TOTAL = %.3f s\n", total);
04011     }
04012 }

```

```

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

```

Write atmospheric data.

Definition at line 4016 of file libtrac.c.

```

04020     {
04021
04022     FILE *in, *out;
04023
04024     char line[LEN];
04025
04026     double r, t0, t1;
04027
04028     int ip, iq, year, mon, day, hour, min, sec;
04029
04030     /* Set timer... */
04031     SELECT_TIMER("WRITE_ATM", NVTX_WRITE);
04032
04033     /* Set time interval for output... */
04034     t0 = t - 0.5 * ctl->dt_mod;
04035     t1 = t + 0.5 * ctl->dt_mod;
04036
04037     /* Write info... */
04038     printf("Write atmospheric data: %s\n", filename);
04039
04040     /* Write ASCII data... */
04041     if (ctl->atm_type == 0) {
04042
04043         /* Check if gnuplot output is requested... */
04044         if (ctl->atm_gpfile[0] != '-') {
04045
04046             /* Create gnuplot pipe... */
04047             if (!(out = popen("gnuplot", "w")))
04048                 ERRMSG("Cannot create pipe to gnuplot!");
04049
04050             /* Set plot filename... */
04051             fprintf(out, "set out \"%s.png\"\n", filename);
04052
04053             /* Set time string... */
04054             jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
04055             fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
04056                     year, mon, day, hour, min);
04057
04058             /* Dump gnuplot file to pipe... */
04059             if (!(in = fopen(ctl->atm_gpfile, "r")))
04060                 ERRMSG("Cannot open file!");
04061             while (fgets(line, LEN, in))
04062                 fprintf(out, "%s", line);
04063             fclose(in);
04064         }
04065
04066     else {
04067
04068         /* Create file... */
04069         if (!(out = fopen(filename, "w")))
04070             ERRMSG("Cannot create file!");
04071     }
04072
04073     /* Write header... */
04074     fprintf(out,
04075            "# $1 = time [s]\n"
04076            "# $2 = altitude [km]\n"
04077            "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
04078     for (iq = 0; iq < ctl->nq; iq++)
04079         fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
04080                ctl->qnt_unit[iq]);
04081     fprintf(out, "\n");
04082
04083     /* Write data... */
04084     for (ip = 0; ip < atm->np; ip += ctl->atm_stride) {
04085
04086         /* Check time... */
04087         if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
04088             continue;
04089
04090         /* Write output... */
04091         fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
04092                atm->lon[ip], atm->lat[ip]);
04093         for (iq = 0; iq < ctl->nq; iq++) {

```

```

04094         fprintf(out, " ");
04095         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
04096     }
04097     fprintf(out, "\n");
04098 }
04099
04100     /* Close file... */
04101     fclose(out);
04102 }
04103
04104     /* Write binary data... */
04105     else if (ctl->atm_type == 1) {
04106
04107         /* Create file... */
04108         if (!(out = fopen(filename, "w")))
04109             ERRMSG("Cannot create file!");
04110
04111         /* Write data... */
04112         FWRITE(&atm->np, int,
04113             1,
04114             out);
04115         FWRITE(atm->time, double,
04116             (size_t) atm->np,
04117             out);
04118         FWRITE(atm->p, double,
04119             (size_t) atm->np,
04120             out);
04121         FWRITE(atm->lon, double,
04122             (size_t) atm->np,
04123             out);
04124         FWRITE(atm->lat, double,
04125             (size_t) atm->np,
04126             out);
04127         for (iq = 0; iq < ctl->nq; iq++)
04128             FWRITE(atm->q[iq], double,
04129                 (size_t) atm->np,
04130                 out);
04131
04132         /* Close file... */
04133         fclose(out);
04134     }
04135
04136     /* Error... */
04137     else
04138         ERRMSG("Atmospheric data type not supported!");
04139 }

```

Here is the call graph for this function:



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

Write CSI data.

Definition at line 4143 of file libtrac.c.

```

04147     {
04148

```

```

04149 static FILE *in, *out;
04150
04151 static char line[LEN];
04152
04153 static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ], rt, rt_old = -1e99,
04154 rz, rlon, rlat, robs, t0, t1, area[GY], dlon, dlat, dz, lat,
04155 x[1000000], y[1000000], work[2000000];
04156
04157 static int obscount[GX][GY][GZ], ct, cx, cy, cz, ip, ix, iy, iz, n;
04158
04159 /* Set timer... */
04160 SELECT_TIMER("WRITE_CSI", NVTX_WRITE);
04161
04162 /* Init... */
04163 if (t == ctl->t_start) {
04164
04165     /* Check quantity index for mass... */
04166     if (ctl->qnt_m < 0)
04167         ERRMSG("Need quantity mass!");
04168
04169     /* Open observation data file... */
04170     printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
04171     if (!(in = fopen(ctl->csi_obsfile, "r")))
04172         ERRMSG("Cannot open file!");
04173
04174     /* Create new file... */
04175     printf("Write CSI data: %s\n", filename);
04176     if (!(out = fopen(filename, "w")))
04177         ERRMSG("Cannot create file!");
04178
04179     /* Write header... */
04180     fprintf(out,
04181         "# $1 = time [s]\n"
04182         "# $2 = number of hits (cx)\n"
04183         "# $3 = number of misses (cy)\n"
04184         "# $4 = number of false alarms (cz)\n"
04185         "# $5 = number of observations (cx + cy)\n"
04186         "# $6 = number of forecasts (cx + cz)\n"
04187         "# $7 = bias (ratio of forecasts and observations) [%%]\n"
04188         "# $8 = probability of detection (POD) [%%]\n"
04189         "# $9 = false alarm rate (FAR) [%%]\n"
04190         "# $10 = critical success index (CSI) [%%]\n");
04191     fprintf(out,
04192         "# $11 = hits associated with random chance\n"
04193         "# $12 = equitable threat score (ETS) [%%]\n"
04194         "# $13 = Pearson linear correlation coefficient\n"
04195         "# $14 = Spearman rank-order correlation coefficient\n"
04196         "# $15 = column density mean error (F - O) [kg/m^2]\n"
04197         "# $16 = column density root mean square error (RMSE) [kg/m^2]\n"
04198         "# $17 = column density mean absolute error [kg/m^2]\n"
04199         "# $18 = number of data points\n\n");
04200
04201     /* Set grid box size... */
04202     dz = (ctl->csi_z1 - ctl->csi_z0) / ctl->csi_nz;
04203     dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
04204     dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
04205
04206     /* Set horizontal coordinates... */
04207     for (iy = 0; iy < ctl->csi_ny; iy++) {
04208         lat = ctl->csi_lat0 + dlat * (iy + 0.5);
04209         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.) * cos(lat * M_PI / 180.);
04210     }
04211 }
04212
04213 /* Set time interval... */
04214 t0 = t - 0.5 * ctl->dt_mod;
04215 t1 = t + 0.5 * ctl->dt_mod;
04216
04217 /* Initialize grid cells... */
04218 #pragma omp parallel for default(shared) private(ix,iy,iz)
04219 for (ix = 0; ix < ctl->csi_nx; ix++)
04220     for (iy = 0; iy < ctl->csi_ny; iy++)
04221         for (iz = 0; iz < ctl->csi_nz; iz++)
04222             modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
04223
04224 /* Read observation data... */
04225 while (fgets(line, LEN, in)) {
04226
04227     /* Read data... */
04228     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
04229         5)
04230         continue;
04231
04232     /* Check time... */
04233     if (rt < t0)
04234         continue;
04235     if (rt > t1)

```

```

04236     break;
04237     if (rt < rt_old)
04238         ERRMSG("Time must be ascending!");
04239     rt_old = rt;
04240
04241     /* Check observation data... */
04242     if (!isfinite(robs))
04243         continue;
04244
04245     /* Calculate indices... */
04246     ix = (int) ((rlon - ctl->csi_lon0) / dlon);
04247     iy = (int) ((rlat - ctl->csi_lat0) / dlat);
04248     iz = (int) ((rz - ctl->csi_z0) / dz);
04249
04250     /* Check indices... */
04251     if (ix < 0 || ix >= ctl->csi_nx ||
04252         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
04253         continue;
04254
04255     /* Get mean observation index... */
04256     obsmean[ix][iy][iz] += robs;
04257     obscount[ix][iy][iz]++;
04258 }
04259
04260 /* Analyze model data... */
04261 for (ip = 0; ip < atm->np; ip++) {
04262
04263     /* Check time... */
04264     if (atm->time[ip] < t0 || atm->time[ip] > t1)
04265         continue;
04266
04267     /* Get indices... */
04268     ix = (int) ((atm->lon[ip] - ctl->csi_lon0) / dlon);
04269     iy = (int) ((atm->lat[ip] - ctl->csi_lat0) / dlat);
04270     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0) / dz);
04271
04272     /* Check indices... */
04273     if (ix < 0 || ix >= ctl->csi_nx ||
04274         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
04275         continue;
04276
04277     /* Get total mass in grid cell... */
04278     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04279 }
04280
04281 /* Analyze all grid cells... */
04282 for (ix = 0; ix < ctl->csi_nx; ix++)
04283     for (iy = 0; iy < ctl->csi_ny; iy++)
04284         for (iz = 0; iz < ctl->csi_nz; iz++) {
04285
04286             /* Calculate mean observation index... */
04287             if (obscount[ix][iy][iz] > 0)
04288                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
04289
04290             /* Calculate column density... */
04291             if (modmean[ix][iy][iz] > 0)
04292                 modmean[ix][iy][iz] /= (1e6 * area[iy]);
04293
04294             /* Calculate CSI... */
04295             if (obscount[ix][iy][iz] > 0) {
04296                 ct++;
04297                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
04298                     modmean[ix][iy][iz] >= ctl->csi_modmin)
04299                     cx++;
04300                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
04301                     modmean[ix][iy][iz] < ctl->csi_modmin)
04302                     cy++;
04303                 else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
04304                     modmean[ix][iy][iz] >= ctl->csi_modmin)
04305                     cz++;
04306             }
04307
04308             /* Save data for other verification statistics... */
04309             if (obscount[ix][iy][iz] > 0
04310                 && (obsmean[ix][iy][iz] >= ctl->csi_obsmin
04311                     || modmean[ix][iy][iz] >= ctl->csi_modmin)) {
04312                 x[n] = modmean[ix][iy][iz];
04313                 y[n] = obsmean[ix][iy][iz];
04314                 if (++n > 1000000)
04315                     ERRMSG("Too many data points to calculate statistics!");
04316             }
04317         }
04318
04319     /* Write output... */
04320     if (fmod(t, ctl->csi_dt_out) == 0) {
04321
04322         /* Calculate verification statistics

```

```

04323     (https://www.cawcr.gov.au/projects/verification/) ... */
04324     int nobs = cx + cy;
04325     int nfor = cx + cz;
04326     double bias = (nobs > 0) ? 100. * nfor / nobs : GSL_NAN;
04327     double pod = (nobs > 0) ? (100. * cx) / nobs : GSL_NAN;
04328     double far = (nfor > 0) ? (100. * cz) / nfor : GSL_NAN;
04329     double csi = (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN;
04330     double cx_rd = (ct > 0) ? (1. * nobs * nfor) / ct : GSL_NAN;
04331     double ets = (cx + cy + cz - cx_rd > 0) ?
04332         (100. * (cx - cx_rd)) / (cx + cy + cz - cx_rd) : GSL_NAN;
04333     double rho_p =
04334         (n > 0) ? gsl_stats_correlation(x, 1, y, 1, (size_t) n) : GSL_NAN;
04335     double rho_s =
04336         (n > 0) ? gsl_stats_spearman(x, 1, y, 1, (size_t) n, work) : GSL_NAN;
04337     for (int i = 0; i < n; i++)
04338         work[i] = x[i] - y[i];
04339     double mean = (n > 0) ? gsl_stats_mean(work, 1, (size_t) n) : GSL_NAN;
04340     double rmse = (n > 0) ? gsl_stats_sd_with_fixed_mean(work, 1, (size_t) n,
04341         0.0) : GSL_NAN;
04342     double absdev =
04343         (n > 0) ? gsl_stats_absdev_m(work, 1, (size_t) n, 0.0) : GSL_NAN;
04344
04345     /* Write... */
04346     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g %g %g %g %g %d\n",
04347         t, cx, cy, cz, nobs, nfor, bias, pod, far, csi, cx_rd, ets,
04348         rho_p, rho_s, mean, rmse, absdev, n);
04349
04350     /* Set counters to zero... */
04351     n = ct = cx = cy = cz = 0;
04352 }
04353
04354 /* Close file... */
04355 if (t == ctl->t_stop)
04356     fclose(out);
04357 }

```

**5.21.2.46 write\_ens()** void write\_ens (

```

    const char * filename,
    ctl_t * ctl,
    atm_t * atm,
    double t )

```

Write ensemble data.

Definition at line 4361 of file libtrac.c.

```

04365     {
04366
04367     static FILE *out;
04368
04369     static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
04370         t0, t1, x[NENS][3], xm[3];
04371
04372     static int ip, iq;
04373
04374     static size_t i, n;
04375
04376     /* Set timer... */
04377     SELECT_TIMER("WRITE_ENS", NVTX_WRITE);
04378
04379     /* Init... */
04380     if (t == ctl->t_start) {
04381
04382         /* Check quantities... */
04383         if (ctl->qnt_ens < 0)
04384             ERRMSG("Missing ensemble IDs!");
04385
04386         /* Create new file... */
04387         printf("Write ensemble data: %s\n", filename);
04388         if (!(out = fopen(filename, "w")))
04389             ERRMSG("Cannot create file!");
04390
04391         /* Write header... */
04392         fprintf(out,
04393             "# $1 = time [s]\n"
04394             "# $2 = altitude [km]\n"
04395             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
04396         for (iq = 0; iq < ctl->nq; iq++)

```

```

04397     fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
04398             ctl->qnt_name[iq], ctl->qnt_unit[iq]);
04399     for (iq = 0; iq < ctl->nq; iq++)
04400         fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
04401             ctl->qnt_name[iq], ctl->qnt_unit[iq]);
04402     fprintf(out, "# $%d = number of members\n", 5 + 2 * ctl->nq);
04403 }
04404
04405 /* Set time interval... */
04406 t0 = t - 0.5 * ctl->dt_mod;
04407 t1 = t + 0.5 * ctl->dt_mod;
04408
04409 /* Init... */
04410 ens = GSL_NAN;
04411 n = 0;
04412
04413 /* Loop over air parcels... */
04414 for (ip = 0; ip < atm->np; ip++) {
04415
04416     /* Check time... */
04417     if (atm->time[ip] < t0 || atm->time[ip] > t1)
04418         continue;
04419
04420     /* Check ensemble id... */
04421     if (atm->q[ctl->qnt_ens][ip] != ens) {
04422
04423         /* Write results... */
04424         if (n > 0) {
04425
04426             /* Get mean position... */
04427             xm[0] = xm[1] = xm[2] = 0;
04428             for (i = 0; i < n; i++) {
04429                 xm[0] += x[i][0] / (double) n;
04430                 xm[1] += x[i][1] / (double) n;
04431                 xm[2] += x[i][2] / (double) n;
04432             }
04433             cart2geo(xm, &dummy, &lon, &lat);
04434             fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
04435                 lat);
04436
04437             /* Get quantity statistics... */
04438             for (iq = 0; iq < ctl->nq; iq++) {
04439                 fprintf(out, " ");
04440                 fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
04441             }
04442             for (iq = 0; iq < ctl->nq; iq++) {
04443                 fprintf(out, " ");
04444                 fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
04445             }
04446             fprintf(out, " %lu\n", n);
04447         }
04448
04449         /* Init new ensemble... */
04450         ens = atm->q[ctl->qnt_ens][ip];
04451         n = 0;
04452     }
04453
04454     /* Save data... */
04455     p[n] = atm->p[ip];
04456     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
04457     for (iq = 0; iq < ctl->nq; iq++)
04458         q[iq][n] = atm->q[iq][ip];
04459     if ((++n) >= NENS)
04460         ERRMSG("Too many data points!");
04461 }
04462
04463 /* Write results... */
04464 if (n > 0) {
04465
04466     /* Get mean position... */
04467     xm[0] = xm[1] = xm[2] = 0;
04468     for (i = 0; i < n; i++) {
04469         xm[0] += x[i][0] / (double) n;
04470         xm[1] += x[i][1] / (double) n;
04471         xm[2] += x[i][2] / (double) n;
04472     }
04473     cart2geo(xm, &dummy, &lon, &lat);
04474     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
04475
04476     /* Get quantity statistics... */
04477     for (iq = 0; iq < ctl->nq; iq++) {
04478         fprintf(out, " ");
04479         fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
04480     }
04481     for (iq = 0; iq < ctl->nq; iq++) {
04482         fprintf(out, " ");
04483         fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));

```

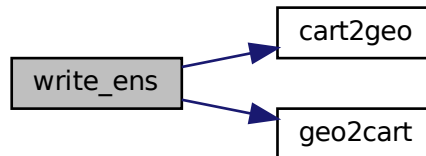


```

04484     }
04485     fprintf(out, " %lu\n", n);
04486 }
04487
04488 /* Close file... */
04489 if (t == ctl->t_stop)
04490     fclose(out);
04491 }

```

Here is the call graph for this function:



**5.21.2.47 write\_grid()** 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 4495 of file libtrac.c.

```

04501     {
04502
04503     FILE *in, *out;
04504
04505     char line[LEN];
04506
04507     static double mass[GX][GY][GZ], z[GZ], dz, lon[GX], dlon, lat[GY], dlat,
04508         area[GY], rho_air, press[GZ], temp, cd, vmr, t0, t1, r;
04509
04510     static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec;
04511
04512     /* Set timer... */
04513     SELECT_TIMER("WRITE_GRID", NVTX_WRITE);
04514
04515     /* Check dimensions... */
04516     if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
04517         ERRMSG("Grid dimensions too large!");
04518
04519     /* Set grid box size... */
04520     dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
04521     dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
04522     dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
04523
04524     /* Set vertical coordinates... */
04525     for (iz = 0; iz < ctl->grid_nz; iz++) {
04526         z[iz] = ctl->grid_z0 + dz * (iz + 0.5);
04527         press[iz] = P(z[iz]);
04528     }
04529
04530     /* Set horizontal coordinates... */
04531     for (ix = 0; ix < ctl->grid_nx; ix++)
04532         lon[ix] = ctl->grid_lon0 + dlon * (ix + 0.5);

```

```

04533     for (iy = 0; iy < ctl->grid_ny; iy++) {
04534         lat[iy] = ctl->grid_lat0 + dlat * (iy + 0.5);
04535         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
04536             * cos(lat[iy] * M_PI / 180.);
04537     }
04538
04539     /* Set time interval for output... */
04540     t0 = t - 0.5 * ctl->dt_mod;
04541     t1 = t + 0.5 * ctl->dt_mod;
04542
04543     /* Initialize grid... */
04544     #pragma omp parallel for default(shared) private(ix,iy,iz)
04545     for (ix = 0; ix < ctl->grid_nx; ix++)
04546         for (iy = 0; iy < ctl->grid_ny; iy++)
04547             for (iz = 0; iz < ctl->grid_nz; iz++) {
04548                 mass[ix][iy][iz] = 0;
04549                 np[ix][iy][iz] = 0;
04550             }
04551
04552     /* Average data... */
04553     for (ip = 0; ip < atm->np; ip++)
04554         if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
04555
04556             /* Get index... */
04557             ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
04558             iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
04559             iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
04560
04561             /* Check indices... */
04562             if (ix < 0 || ix >= ctl->grid_nx ||
04563                 iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
04564                 continue;
04565
04566             /* Add mass... */
04567             if (ctl->qnt_m >= 0)
04568                 mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04569             np[ix][iy][iz]++;
04570         }
04571
04572     /* Check if gnuplot output is requested... */
04573     if (ctl->grid_gpfile[0] != '-') {
04574
04575         /* Write info... */
04576         printf("Plot grid data: %s.png\n", filename);
04577
04578         /* Create gnuplot pipe... */
04579         if (!(out = popen("gnuplot", "w")))
04580             ERRMSG("Cannot create pipe to gnuplot!");
04581
04582         /* Set plot filename... */
04583         fprintf(out, "set out \"%s.png\"\\n", filename);
04584
04585         /* Set time string... */
04586         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
04587         fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\\n",
04588             year, mon, day, hour, min);
04589
04590         /* Dump gnuplot file to pipe... */
04591         if (!(in = fopen(ctl->grid_gpfile, "r")))
04592             ERRMSG("Cannot open file!");
04593         while (fgets(line, LEN, in))
04594             fprintf(out, "%s", line);
04595         fclose(in);
04596     }
04597
04598     else {
04599
04600         /* Write info... */
04601         printf("Write grid data: %s\\n", filename);
04602
04603         /* Create file... */
04604         if (!(out = fopen(filename, "w")))
04605             ERRMSG("Cannot create file!");
04606     }
04607
04608     /* Write header... */
04609     fprintf(out,
04610         "# $1 = time [s]\\n"
04611         "# $2 = altitude [km]\\n"
04612         "# $3 = longitude [deg]\\n"
04613         "# $4 = latitude [deg]\\n"
04614         "# $5 = surface area [km^2]\\n"
04615         "# $6 = layer width [km]\\n"
04616         "# $7 = number of particles [l]\\n"
04617         "# $8 = column density [kg/m^2]\\n"
04618         "# $9 = volume mixing ratio [ppv]\\n\\n");
04619

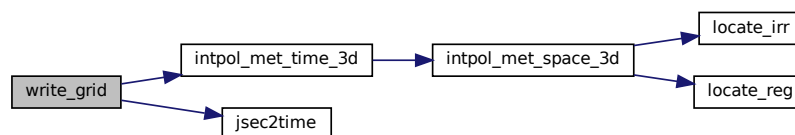
```

```

04620  /* Write data... */
04621  for (ix = 0; ix < ctl->grid_nx; ix++) {
04622      if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
04623          fprintf(out, "\n");
04624      for (iy = 0; iy < ctl->grid_ny; iy++) {
04625          if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
04626              fprintf(out, "\n");
04627          for (iz = 0; iz < ctl->grid_nz; iz++)
04628              if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
04629
04630                  /* Calculate column density... */
04631                  cd = mass[ix][iy][iz] / (1e6 * area[iy]);
04632
04633                  /* Calculate volume mixing ratio... */
04634                  vmr = 0;
04635                  if (ctl->molmass > 0) {
04636                      if (mass[ix][iy][iz] > 0) {
04637
04638                          /* Get temperature... */
04639                          INTPOL_INIT;
04640                          intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
04641                                              lon[ix], lat[iy], &temp, ci, cw, 1);
04642
04643                          /* Calculate density of air... */
04644                          rho_air = 100. * press[iz] / (RA * temp);
04645
04646                          /* Calculate volume mixing ratio... */
04647                          vmr = MA / ctl->molmass * mass[ix][iy][iz]
04648                              / (rho_air * 1e6 * area[iy] * 1e3 * dz);
04649                      }
04650                  } else
04651                      vmr = GSL_NAN;
04652
04653                  /* Write output... */
04654                  fprintf(out, "%.2f %g %g %g %g %d %g %g\n", t, z[iz],
04655                          lon[ix], lat[iy], area[iy], dz, np[ix][iy][iz], cd, vmr);
04656              }
04657          }
04658      }
04659
04660      /* Close file... */
04661      fclose(out);
04662  }

```

Here is the call graph for this function:



**5.21.2.48 write\_prof()** 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 4666 of file libtrac.c.

```

04672     {
04673

```

```

04674 static FILE *in, *out;
04675
04676 static char line[LEN];
04677
04678 static double mass[GX][GY][GZ], obsmean[GX][GY], rt, rt_old = -1e99,
04679 rz, rlon, rlat, robs, t0, t1, area[GY], dz, dlon, dlat, lon[GX], lat[GY],
04680 z[GZ], press[GZ], temp, rho_air, vmr, h2o, o3;
04681
04682 static int obscount[GX][GY], ip, ix, iy, iz, okay;
04683
04684 /* Set timer... */
04685 SELECT_TIMER("WRITE_PROF", NVTX_WRITE);
04686
04687 /* Init... */
04688 if (t == ctl->t_start) {
04689
04690     /* Check quantity index for mass... */
04691     if (ctl->qnt_m < 0)
04692         ERRMSG("Need quantity mass!");
04693
04694     /* Check dimensions... */
04695     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
04696         ERRMSG("Grid dimensions too large!");
04697
04698     /* Check molar mass... */
04699     if (ctl->molmass <= 0)
04700         ERRMSG("Specify molar mass!");
04701
04702     /* Open observation data file... */
04703     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
04704     if (!(in = fopen(ctl->prof_obsfile, "r")))
04705         ERRMSG("Cannot open file!");
04706
04707     /* Create new output file... */
04708     printf("Write profile data: %s\n", filename);
04709     if (!(out = fopen(filename, "w")))
04710         ERRMSG("Cannot create file!");
04711
04712     /* Write header... */
04713     fprintf(out,
04714         "# $1 = time [s]\n"
04715         "# $2 = altitude [km]\n"
04716         "# $3 = longitude [deg]\n"
04717         "# $4 = latitude [deg]\n"
04718         "# $5 = pressure [hPa]\n"
04719         "# $6 = temperature [K]\n"
04720         "# $7 = volume mixing ratio [ppv]\n"
04721         "# $8 = H2O volume mixing ratio [ppv]\n"
04722         "# $9 = O3 volume mixing ratio [ppv]\n"
04723         "# $10 = observed BT index [K]\n");
04724
04725     /* Set grid box size... */
04726     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
04727     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
04728     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
04729
04730     /* Set vertical coordinates... */
04731     for (iz = 0; iz < ctl->prof_nz; iz++) {
04732         z[iz] = ctl->prof_z0 + dz * (iz + 0.5);
04733         press[iz] = P(z[iz]);
04734     }
04735
04736     /* Set horizontal coordinates... */
04737     for (ix = 0; ix < ctl->prof_nx; ix++)
04738         lon[ix] = ctl->prof_lon0 + dlon * (ix + 0.5);
04739     for (iy = 0; iy < ctl->prof_ny; iy++) {
04740         lat[iy] = ctl->prof_lat0 + dlat * (iy + 0.5);
04741         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
04742             * cos(lat[iy] * M_PI / 180.);
04743     }
04744 }
04745
04746 /* Set time interval... */
04747 t0 = t - 0.5 * ctl->dt_mod;
04748 t1 = t + 0.5 * ctl->dt_mod;
04749
04750 /* Initialize... */
04751 #pragma omp parallel for default(shared) private(ix,iy,iz)
04752 for (ix = 0; ix < ctl->prof_nx; ix++)
04753     for (iy = 0; iy < ctl->prof_ny; iy++) {
04754         obsmean[ix][iy] = 0;
04755         obscount[ix][iy] = 0;
04756         for (iz = 0; iz < ctl->prof_nz; iz++)
04757             mass[ix][iy][iz] = 0;
04758     }
04759
04760 /* Read observation data... */

```

```

04761 while (fgets(line, LEN, in)) {
04762
04763     /* Read data... */
04764     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &robs) !=
04765         5)
04766         continue;
04767
04768     /* Check time... */
04769     if (rt < t0)
04770         continue;
04771     if (rt > t1)
04772         break;
04773     if (rt < rt_old)
04774         ERRMSG("Time must be ascending!");
04775     rt_old = rt;
04776
04777     /* Check observation data... */
04778     if (!isfinite(robs))
04779         continue;
04780
04781     /* Calculate indices... */
04782     ix = (int) ((rln - ctl->prof_lon0) / dlon);
04783     iy = (int) ((rln - ctl->prof_lat0) / dlat);
04784
04785     /* Check indices... */
04786     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
04787         continue;
04788
04789     /* Get mean observation index... */
04790     obsmean[ix][iy] += robs;
04791     obscount[ix][iy]++;
04792 }
04793
04794 /* Analyze model data... */
04795 for (ip = 0; ip < atm->np; ip++) {
04796
04797     /* Check time... */
04798     if (atm->time[ip] < t0 || atm->time[ip] > t1)
04799         continue;
04800
04801     /* Get indices... */
04802     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
04803     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
04804     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
04805
04806     /* Check indices... */
04807     if (ix < 0 || ix >= ctl->prof_nx ||
04808         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
04809         continue;
04810
04811     /* Get total mass in grid cell... */
04812     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04813 }
04814
04815 /* Extract profiles... */
04816 for (ix = 0; ix < ctl->prof_nx; ix++)
04817     for (iy = 0; iy < ctl->prof_ny; iy++)
04818         if (obscount[ix][iy] > 0) {
04819
04820             /* Check profile... */
04821             okay = 0;
04822             for (iz = 0; iz < ctl->prof_nz; iz++)
04823                 if (mass[ix][iy][iz] > 0) {
04824                     okay = 1;
04825                     break;
04826                 }
04827             if (!okay)
04828                 continue;
04829
04830             /* Write output... */
04831             fprintf(out, "\n");
04832
04833             /* Loop over altitudes... */
04834             for (iz = 0; iz < ctl->prof_nz; iz++) {
04835
04836                 /* Get pressure and temperature... */
04837                 INTPOL_INIT;
04838                 intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
04839                                     lon[ix], lat[iy], &temp, ci, cw, 1);
04840                 intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, t, press[iz],
04841                                     lon[ix], lat[iy], &h2o, ci, cw, 0);
04842                 intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press[iz],
04843                                     lon[ix], lat[iy], &o3, ci, cw, 0);
04844
04845                 /* Calculate volume mixing ratio... */
04846                 rho_air = 100. * press[iz] / (RA * temp);
04847                 vmr = MA / ctl->molmass * mass[ix][iy][iz]

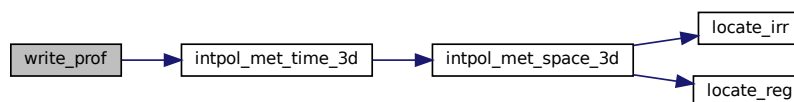
```

```

04848         / (rho_air * area[iy] * dz * 1e9);
04849
04850     /* Write output... */
04851     fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
04852            t, z[iz], lon[ix], lat[iy], press[iz], temp, vmr, h2o, o3,
04853            obsmean[ix][iy] / obscount[ix][iy]);
04854     }
04855 }
04856
04857 /* Close files... */
04858 if (t == ctl->t_stop) {
04859     fclose(in);
04860     fclose(out);
04861 }
04862 }

```

Here is the call graph for this function:



**5.21.2.49 write\_sample()** void write\_sample (

```

    const char * filename,
    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double t )

```

Write sample data.

Definition at line 4866 of file libtrac.c.

```

04872     {
04873
04874     static FILE *in, *out;
04875
04876     static char line[LEN];
04877
04878     static double area, dlat, rmax2, t0, t1, rt, rt_old, rz, rlon, rlat, robs;
04879
04880     /* Set timer... */
04881     SELECT_TIMER("WRITE_SAMPLE", NVTX_WRITE);
04882
04883     /* Init... */
04884     if (t == ctl->t_start) {
04885
04886         /* Open observation data file... */
04887         printf("Read sample observation data: %s\n", ctl->sample_obsfile);
04888         if (!(in = fopen(ctl->sample_obsfile, "r")))
04889             ERRMSG("Cannot open file!");
04890
04891         /* Create new file... */
04892         printf("Write sample data: %s\n", filename);
04893         if (!(out = fopen(filename, "w")))
04894             ERRMSG("Cannot create file!");
04895
04896         /* Write header... */
04897         fprintf(out,
04898             "# $1 = time [s]\n"
04899             "# $2 = altitude [km]\n"
04900             "# $3 = longitude [deg]\n"
04901             "# $4 = latitude [deg]\n"
04902             "# $5 = surface area [km^2]\n"

```

```

04903         "# $6 = layer width [km]\n"
04904         "# $7 = number of particles [l]\n"
04905         "# $8 = column density [kg/m^2]\n"
04906         "# $9 = volume mixing ratio [ppv]\n"
04907         "# $10 = observed BT index [K]\n\n");
04908
04909     /* Set latitude range, squared radius, and area... */
04910     dlat = DY2DEG(ctl->sample_dx);
04911     rmax2 = SQR(ctl->sample_dx);
04912     area = M_PI * rmax2;
04913 }
04914
04915 /* Set time interval for output... */
04916 t0 = t - 0.5 * ctl->dt_mod;
04917 t1 = t + 0.5 * ctl->dt_mod;
04918
04919 /* Read observation data... */
04920 while (fgets(line, LEN, in)) {
04921
04922     /* Read data... */
04923     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &rln) !=
04924         5)
04925         continue;
04926
04927     /* Check time... */
04928     if (rt < t0)
04929         continue;
04930     if (rt < rt_old)
04931         ERRMSG("Time must be ascending!");
04932     rt_old = rt;
04933
04934     /* Calculate Cartesian coordinates... */
04935     double x0[3];
04936     geo2cart(0, rln, rln, x0);
04937
04938     /* Set pressure range... */
04939     double rp = P(rz);
04940     double ptop = P(rz + ctl->sample_dz);
04941     double pbot = P(rz - ctl->sample_dz);
04942
04943     /* Init... */
04944     double mass = 0;
04945     int np = 0;
04946
04947     /* Loop over air parcels... */
04948 #pragma omp parallel for default(shared) reduction(+:mass,np)
04949     for (int ip = 0; ip < atm->np; ip++) {
04950
04951         /* Check time... */
04952         if (atm->time[ip] < t0 || atm->time[ip] > t1)
04953             continue;
04954
04955         /* Check latitude... */
04956         if (fabs(rln - atm->lat[ip]) > dlat)
04957             continue;
04958
04959         /* Check horizontal distance... */
04960         double x1[3];
04961         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
04962         if (DIST2(x0, x1) > rmax2)
04963             continue;
04964
04965         /* Check pressure... */
04966         if (ctl->sample_dz > 0)
04967             if (atm->p[ip] > pbot || atm->p[ip] < ptop)
04968                 continue;
04969
04970         /* Add mass... */
04971         if (ctl->qnt_m >= 0)
04972             mass += atm->q[ctl->qnt_m][ip];
04973         np++;
04974     }
04975
04976     /* Calculate column density... */
04977     double cd = mass / (1e6 * area);
04978
04979     /* Calculate volume mixing ratio... */
04980     double vmr = 0;
04981     if (ctl->molmass > 0 && ctl->sample_dz > 0) {
04982         if (mass > 0) {
04983
04984             /* Get temperature... */
04985             double temp;
04986             INTPOL_INIT;
04987             intpol_met_time_3d(met0, met0->t, met1, met1->t, rt, rp,
04988                             rln, rln, &temp, ci, cw, 1);
04989

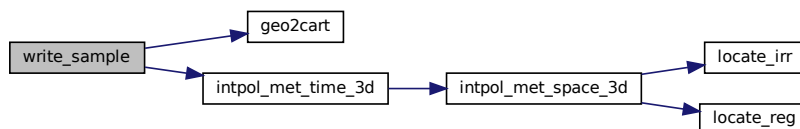
```

```

04990      /* Calculate density of air... */
04991      double rho_air = 100. * rp / (RA * temp);
04992
04993      /* Calculate volume mixing ratio... */
04994      vmr = MA / ctl->molmass * mass
04995            / (rho_air * 1e6 * area * 1e3 * ctl->sample_dz);
04996    }
04997  } else
04998    vmr = GSL_NAN;
04999
05000  /* Write output... */
05001  fprintf(out, "%.2f %g %g %g %g %d %g %g %g\n", rt, rz, rlon, rlat,
05002          area, ctl->sample_dz, np, cd, vmr, robs);
05003
05004  /* Check time... */
05005  if (rt >= t1)
05006    break;
05007  }
05008
05009  /* Close files... */
05010  if (t == ctl->t_stop) {
05011    fclose(in);
05012    fclose(out);
05013  }
05014 }

```

Here is the call graph for this function:



**5.21.2.50 write\_station()** void write\_station (

```

    const char * filename,
    ctl_t * ctl,
    atm_t * atm,
    double t )

```

Write station data.

Definition at line 5018 of file libtrac.c.

```

05022      {
05023
05024      static FILE *out;
05025
05026      static double rmax2, t0, t1, x0[3], x1[3];
05027
05028      /* Set timer... */
05029      SELECT_TIMER("WRITE_STATION", NVTX_WRITE);
05030
05031      /* Init... */
05032      if (t == ctl->t_start) {
05033
05034          /* Write info... */
05035          printf("Write station data: %s\n", filename);
05036
05037          /* Create new file... */
05038          if (!(out = fopen(filename, "w")))
05039              ERRMSG("Cannot create file!");
05040
05041          /* Write header... */
05042          fprintf(out,
05043                  "# $1 = time [s]\n"
05044                  "# $2 = altitude [km]\n"

```



```

05045         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
05046     for (int iq = 0; iq < ctl->nq; iq++)
05047         fprintf(out, "# $i = %s [%s]\n", (iq + 5),
05048             ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05049     fprintf(out, "\n");
05050
05051     /* Set geolocation and search radius... */
05052     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
05053     rmax2 = SQR(ctl->stat_r);
05054 }
05055
05056 /* Set time interval for output... */
05057 t0 = t - 0.5 * ctl->dt_mod;
05058 t1 = t + 0.5 * ctl->dt_mod;
05059
05060 /* Loop over air parcels... */
05061 for (int ip = 0; ip < atm->np; ip++) {
05062
05063     /* Check time... */
05064     if (atm->time[ip] < t0 || atm->time[ip] > t1)
05065         continue;
05066
05067     /* Check station flag... */
05068     if (ctl->qnt_stat >= 0)
05069         if (atm->q[ctl->qnt_stat][ip])
05070             continue;
05071
05072     /* Get Cartesian coordinates... */
05073     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
05074
05075     /* Check horizontal distance... */
05076     if (DIST2(x0, x1) > rmax2)
05077         continue;
05078
05079     /* Set station flag... */
05080     if (ctl->qnt_stat >= 0)
05081         atm->q[ctl->qnt_stat][ip] = 1;
05082
05083     /* Write data... */
05084     fprintf(out, "%.2f %g %g %g",
05085         atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
05086     for (int iq = 0; iq < ctl->nq; iq++) {
05087         fprintf(out, " ");
05088         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
05089     }
05090     fprintf(out, "\n");
05091 }
05092
05093 /* Close file... */
05094 if (t == ctl->t_stop)
05095     fclose(out);
05096 }

```

Here is the call graph for this function:



## 5.22 libtrac.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
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /*****
00028
00029 void cart2geo(
00030     double *x,
00031     double *z,
00032     double *lon,
00033     double *lat) {
00034
00035     double radius = NORM(x);
00036     *lat = asin(x[2] / radius) * 180. / M_PI;
00037     *lon = atan2(x[1], x[0]) * 180. / M_PI;
00038     *z = radius - RE;
00039 }
00040
00041 /*****
00042
00043 static double clim_hno3_secs[12] = {
00044     1209600.00, 3888000.00, 6393600.00,
00045     9072000.00, 11664000.00, 14342400.00,
00046     16934400.00, 19612800.00, 22291200.00,
00047     24883200.00, 27561600.00, 30153600.00
00048 };
00049
00050 #ifdef _OPENACC
00051 #pragma acc declare copyin(clim_hno3_secs)
00052 #endif
00053
00054 static double clim_hno3_lats[18] = {
00055     -85, -75, -65, -55, -45, -35, -25, -15, -5,
00056     5, 15, 25, 35, 45, 55, 65, 75, 85
00057 };
00058
00059 #ifdef _OPENACC
00060 #pragma acc declare copyin(clim_hno3_lats)
00061 #endif
00062
00063 static double clim_hno3_ps[10] = {
00064     4.64159, 6.81292, 10, 14.678, 21.5443,
00065     31.6228, 46.4159, 68.1292, 100, 146.78
00066 };
00067
00068 #ifdef _OPENACC
00069 #pragma acc declare copyin(clim_hno3_ps)
00070 #endif
00071
00072 static double clim_hno3_var[12][18][10] = {
00073     {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00074      {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00075      {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
00076      {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00077      {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
00078      {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
00079      {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00080      {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
00081      {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
00082      {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00083      {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
00084      {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00085      {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
00086      {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00087      {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00088      {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00089      {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
00090      {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
00091     {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00092      {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
00093      {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
00094      {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00095      {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
00096      {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
00097      {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
00098      {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
00099      {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
00100      {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
00101      {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00102      {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
00103      {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},

```

```
00104 {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
00105 {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
00106 {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
00107 {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
00108 {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17},
00109 {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00110 {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
00111 {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
00112 {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
00113 {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
00114 {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
00115 {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
00116 {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
00117 {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
00118 {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
00119 {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
00120 {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
00121 {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
00122 {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
00123 {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
00124 {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
00125 {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
00126 {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42},
00127 {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
00128 {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00129 {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
00130 {1.28, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
00131 {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
00132 {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
00133 {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
00134 {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
00135 {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
00136 {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
00137 {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
00138 {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
00139 {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
00140 {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
00141 {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
00142 {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
00143 {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
00144 {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62},
00145 {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
00146 {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
00147 {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
00148 {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
00149 {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
00150 {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
00151 {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
00152 {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
00153 {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
00154 {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
00155 {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
00156 {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
00157 {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
00158 {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
00159 {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
00160 {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
00161 {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
00162 {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6},
00163 {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
00164 {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
00165 {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
00166 {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
00167 {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
00168 {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
00169 {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
00170 {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
00171 {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
00172 {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
00173 {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
00174 {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
00175 {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
00176 {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
00177 {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
00178 {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
00179 {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
00180 {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91},
00181 {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
00182 {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
00183 {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
00184 {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
00185 {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
00186 {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
00187 {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
00188 {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
00189 {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
00190 {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
```

```

00191 {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
00192 {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
00193 {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
00194 {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
00195 {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
00196 {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
00197 {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
00198 {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62},
00199 {5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
00200 {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
00201 {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
00202 {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
00203 {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
00204 {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
00205 {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
00206 {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
00207 {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
00208 {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
00209 {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
00210 {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
00211 {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
00212 {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
00213 {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
00214 {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
00215 {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
00216 {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55},
00217 {1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
00218 {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
00219 {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
00220 {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
00221 {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
00222 {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
00223 {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
00224 {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
00225 {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
00226 {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
00227 {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
00228 {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
00229 {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
00230 {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
00231 {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
00232 {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
00233 {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
00234 {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65},
00235 {0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
00236 {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
00237 {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
00238 {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
00239 {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
00240 {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
00241 {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
00242 {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
00243 {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
00244 {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
00245 {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
00246 {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
00247 {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
00248 {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
00249 {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
00250 {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
00251 {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
00252 {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8},
00253 {0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
00254 {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
00255 {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
00256 {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
00257 {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
00258 {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
00259 {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
00260 {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
00261 {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
00262 {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
00263 {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
00264 {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
00265 {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
00266 {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
00267 {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
00268 {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
00269 {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
00270 {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05},
00271 {0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
00272 {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
00273 {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
00274 {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
00275 {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
00276 {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
00277 {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},

```

```

00278     {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
00279     {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
00280     {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
00281     {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
00282     {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
00283     {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
00284     {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
00285     {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
00286     {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
00287     {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
00288     {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
00289 };
00290
00291 #ifdef _OPENACC
00292 #pragma acc declare copyin(clim_hno3_var)
00293 #endif
00294
00295 double clim_hno3(
00296     double t,
00297     double lat,
00298     double p) {
00299
00300     /* Get seconds since begin of year... */
00301     double sec = FMOD(t, 365.25 * 86400.);
00302     while (sec < 0)
00303         sec += 365.25 * 86400.;
00304
00305     /* Check pressure... */
00306     if (p < clim_hno3_ps[0])
00307         p = clim_hno3_ps[0];
00308     else if (p > clim_hno3_ps[9])
00309         p = clim_hno3_ps[9];
00310
00311     /* Check latitude... */
00312     if (lat < clim_hno3_lats[0])
00313         lat = clim_hno3_lats[0];
00314     else if (lat > clim_hno3_lats[17])
00315         lat = clim_hno3_lats[17];
00316
00317     /* Get indices... */
00318     int isec = locate_irr(clim_hno3_secs, 12, sec);
00319     int ilat = locate_reg(clim_hno3_lats, 18, lat);
00320     int ip = locate_irr(clim_hno3_ps, 10, p);
00321
00322     /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
00323     double aux00 = LIN(clim_hno3_ps[ip],
00324         clim_hno3_var[isec][ilat][ip],
00325         clim_hno3_ps[ip + 1],
00326         clim_hno3_var[isec][ilat][ip + 1], p);
00327     double aux01 = LIN(clim_hno3_ps[ip],
00328         clim_hno3_var[isec][ilat + 1][ip],
00329         clim_hno3_ps[ip + 1],
00330         clim_hno3_var[isec][ilat + 1][ip + 1], p);
00331     double aux10 = LIN(clim_hno3_ps[ip],
00332         clim_hno3_var[isec + 1][ilat][ip],
00333         clim_hno3_ps[ip + 1],
00334         clim_hno3_var[isec + 1][ilat][ip + 1], p);
00335     double aux11 = LIN(clim_hno3_ps[ip],
00336         clim_hno3_var[isec + 1][ilat + 1][ip],
00337         clim_hno3_ps[ip + 1],
00338         clim_hno3_var[isec + 1][ilat + 1][ip + 1], p);
00339     aux00 = LIN(clim_hno3_lats[ilat], aux00,
00340         clim_hno3_lats[ilat + 1], aux01, lat);
00341     aux11 = LIN(clim_hno3_lats[ilat], aux10,
00342         clim_hno3_lats[ilat + 1], aux11, lat);
00343     aux00 = LIN(clim_hno3_secs[isec], aux00,
00344         clim_hno3_secs[isec + 1], aux11, sec);
00345     return GSL_MAX(aux00, 0.0);
00346 }
00347
00348 /*****
00349
00350 static double clim_oh_secs[12] = {
00351     1209600.00, 3888000.00, 6393600.00,
00352     9072000.00, 11664000.00, 14342400.00,
00353     16934400.00, 19612800.00, 22291200.00,
00354     24883200.00, 27561600.00, 30153600.00
00355 };
00356
00357 #ifdef _OPENACC
00358 #pragma acc declare copyin(clim_oh_secs)
00359 #endif
00360
00361 static double clim_oh_lats[18] = {
00362     -85, -75, -65, -55, -45, -35, -25, -15, -5,
00363     5, 15, 25, 35, 45, 55, 65, 75, 85
00364 };

```

```
00365
00366 #ifndef _OPENACC
00367 #pragma acc declare copyin(clim_oh_lats)
00368 #endif
00369
00370 static double clim_oh_ps[34] = {
00371     0.17501, 0.233347, 0.31113, 0.41484, 0.553119, 0.737493, 0.983323,
00372     1.3111, 1.74813, 2.33084, 3.10779, 4.14372, 5.52496, 7.36661, 9.82214,
00373     13.0962, 17.4616, 23.2821, 31.0428, 41.3904, 55.1872, 73.583, 98.1107,
00374     130.814, 174.419, 232.559, 310.078, 413.438, 551.25, 735, 789.809,
00375     848.705, 911.993, 980
00376 };
00377
00378 #ifndef _OPENACC
00379 #pragma acc declare copyin(clim_oh_ps)
00380 #endif
00381
00382 static double clim_oh_var[12][18][34] = {
00383     {{6.422, 6.418, 7.221, 8.409, 9.768, 11.22, 12.65, 13.68, 14.03,
00384       13.06, 11.01, 8.791, 7.096, 6.025, 5.135, 4.057, 2.791, 1.902,
00385       1.318, 0.9553, 0.7083, 0.5542, 0.5145, 0.5485, 0.6292, 0.5982, 1.716,
00386       1.111, 0.9802, 0.6707, 0.5235, 0.4476, 0.3783, 0.3091}},
00387     {{6.311, 6.394, 7.2, 8.349, 9.664, 11.02, 12.21, 13.06, 13.28,
00388       12.42, 10.59, 8.552, 6.944, 5.862, 4.948, 3.826, 2.689, 1.873,
00389       1.302, 0.9316, 0.7053, 0.5634, 0.508, 0.5207, 0.6166, 0.6789, 1.682,
00390       1.218, 1.079, 0.7621, 0.6662, 0.5778, 0.4875, 0.3997}},
00391     {{5.851, 5.827, 6.393, 7.294, 8.322, 9.415, 10.46, 11.24, 11.59,
00392       11.13, 9.754, 7.97, 6.417, 5.331, 4.468, 3.512, 2.581, 1.855,
00393       1.336, 0.9811, 0.756, 0.6328, 0.6011, 0.6202, 0.7603, 0.8883, 1.303,
00394       1.124, 1.118, 0.9428, 0.8655, 0.8156, 0.7602, 0.6805}},
00395     {{5.276, 5.158, 5.66, 6.463, 7.419, 8.488, 9.563, 10.45, 10.94,
00396       10.65, 9.465, 7.762, 6.204, 5.074, 4.209, 3.324, 2.511, 1.865,
00397       1.386, 1.066, 0.8521, 0.723, 0.6997, 0.7492, 0.8705, 0.8088, 1.22,
00398       1.192, 1.298, 1.096, 1.037, 0.9589, 0.8856, 0.7726}},
00399     {{5.06, 4.919, 5.379, 6.142, 7.095, 8.156, 9.18, 10.09, 10.62,
00400       10.33, 9.123, 7.479, 5.967, 4.858, 3.987, 3.097, 2.342, 1.743,
00401       1.323, 1.044, 0.8598, 0.7596, 0.7701, 0.7858, 0.8741, 1.256, 1.266,
00402       1.418, 1.594, 1.247, 1.169, 1.111, 1.054, 0.9141}},
00403     {{4.921, 4.759, 5.188, 5.936, 6.847, 7.871, 8.903, 9.805, 10.31,
00404       10, 8.818, 7.223, 5.757, 4.66, 3.75, 2.831, 2.1, 1.579,
00405       1.243, 1.017, 0.8801, 0.8193, 0.9409, 1.131, 0.7313, 1.201, 1.383,
00406       1.643, 1.751, 1.494, 1.499, 1.647, 1.934, 2.147}},
00407     {{4.665, 4.507, 4.947, 5.652, 6.549, 7.573, 8.609, 9.499, 9.985,
00408       9.664, 8.478, 6.944, 5.519, 4.407, 3.511, 2.595, 1.917, 1.46,
00409       1.172, 1.009, 0.9372, 0.9439, 1.047, 1.219, 0.5712, 1.032, 1.342,
00410       1.716, 1.846, 1.551, 1.55, 1.686, 2.006, 2.235}},
00411     {{4.424, 4.288, 4.678, 5.38, 6.271, 7.291, 8.324, 9.231, 9.678,
00412       9.264, 8.037, 6.532, 5.141, 4.037, 3.148, 2.319, 1.715, 1.318,
00413       1.078, 0.9647, 0.9327, 0.9604, 1.023, 0.4157, 0.4762, 1.04, 1.589,
00414       2.093, 1.957, 1.557, 1.52, 1.565, 1.776, 1.904}},
00415     {{4.154, 3.996, 4.347, 5.004, 5.854, 6.869, 7.929, 8.837, 9.23,
00416       8.708, 7.447, 6.024, 4.761, 3.742, 2.898, 2.096, 1.55, 1.191,
00417       0.9749, 0.8889, 0.8745, 0.9004, 0.9648, 0.36, 0.4423, 0.973, 1.571,
00418       2.086, 1.971, 1.569, 1.537, 1.567, 1.74, 1.811}},
00419     {{3.862, 3.738, 4.093, 4.693, 5.499, 6.481, 7.489, 8.328, 8.637,
00420       8.07, 6.863, 5.56, 4.438, 3.522, 2.736, 1.971, 1.441, 1.098,
00421       0.8945, 0.8155, 0.7965, 0.8013, 0.8582, 1.119, 0.4076, 0.8805, 1.446,
00422       1.977, 1.96, 1.713, 1.793, 2.055, 2.521, 2.776}},
00423     {{3.619, 3.567, 3.943, 4.54, 5.295, 6.168, 7.033, 7.691, 7.884,
00424       7.326, 6.207, 5.032, 4.055, 3.263, 2.552, 1.871, 1.365, 1.022,
00425       0.8208, 0.7184, 0.6701, 0.6551, 0.6965, 0.7928, 0.3639, 0.6365, 0.9295,
00426       1.381, 1.847, 1.658, 1.668, 1.87, 2.245, 2.409}},
00427     {{3.354, 3.395, 3.811, 4.39, 5.07, 5.809, 6.514, 7, 7.054,
00428       6.472, 5.463, 4.466, 3.649, 2.997, 2.396, 1.785, 1.289, 0.9304,
00429       0.7095, 0.5806, 0.5049, 0.4639, 0.4899, 0.5149, 0.5445, 0.5185, 0.7495,
00430       0.8662, 1.25, 1.372, 1.384, 1.479, 1.76, 1.874}},
00431     {{3.008, 3.102, 3.503, 4.049, 4.657, 5.287, 5.845, 6.14, 6.032,
00432       5.401, 4.494, 3.665, 3.043, 2.575, 2.103, 1.545, 1.074, 0.7429,
00433       0.5514, 0.4313, 0.3505, 0.2957, 0.2688, 0.2455, 0.232, 0.3565, 0.4017,
00434       0.5063, 0.6618, 0.7621, 0.7915, 0.8372, 0.923, 0.9218}},
00435     {{2.548, 2.725, 3.135, 3.637, 4.165, 4.666, 5.013, 5.056, 4.72,
00436       4.033, 3.255, 2.64, 2.24, 1.942, 1.555, 1.085, 0.7271, 0.502,
00437       0.3748, 0.2897, 0.2303, 0.19, 0.1645, 0.1431, 0.1215, 0.09467, 0.1442,
00438       0.1847, 0.2368, 0.2463, 0.2387, 0.2459, 0.2706, 0.2751}},
00439     {{1.946, 2.135, 2.46, 2.831, 3.203, 3.504, 3.584, 3.37, 2.921,
00440       2.357, 1.865, 1.551, 1.392, 1.165, 0.8443, 0.5497, 0.3686, 0.2632,
00441       0.1978, 0.1509, 0.1197, 0.0992, 0.08402, 0.07068, 0.05652, 0.03962,
00442       0.03904,
00443       0.04357, 0.05302, 0.04795, 0.04441, 0.04296, 0.04446, 0.04576}},
00444     {{1.157, 1.285, 1.432, 1.546, 1.602, 1.556, 1.378, 1.15, 0.9351,
00445       0.7636, 0.6384, 0.5267, 0.4008, 0.2821, 0.2, 0.1336, 0.09109, 0.06557,
00446       0.05219, 0.04197, 0.03443, 0.03119, 0.02893, 0.02577, 0.02119, 0.01102,
00447       0.008897,
00448       0.00467, 0.004651, 0.004809, 0.004539, 0.004181, 0.003737, 0.002833}},
00449     {{0.07716, 0.06347, 0.05343, 0.04653, 0.0393, 0.03205, 0.02438, 0.01692,
00450       0.0115,
00451       0.007576, 0.00488, 0.002961, 0.001599, 0.001033, 0.001067, 0.001091,
```

00452 0.0005156, 0.0003818,  
00453 0.0005061, 0.0005322, 0.0008027, 0.0008598, 0.0009114, 0.001112, 0.002042,  
00454 0.0002528, 0.0005562,  
00455 7.905e-06, 1.461e-07, 1.448e-07, 9.962e-08, 4.304e-08, 9.129e-17,  
00456 1.36e-16},  
00457 {2.613e-05, 3.434e-05, 3.646e-05, 5.101e-05, 8.027e-05, 0.0001172,  
00458 9.886e-05, 1.933e-05, 3.14e-05,  
00459 7.708e-05, 0.000136, 0.0001447, 0.0001049, 4.451e-05, 9.37e-06, 2.235e-06,  
00460 1.034e-06, 4.87e-06,  
00461 1.615e-05, 2.018e-05, 6.578e-05, 0.000178, 0.0002489, 0.0004818, 0.001231,  
00462 0.0001402, 0.0004263,  
00463 4.581e-07, 1.045e-12, 1.295e-13, 9.008e-14, 8.464e-14, 1.183e-13,  
00464 2.471e-13}},  
00465 {5.459, 5.793, 6.743, 7.964, 9.289, 10.5, 11.39, 11.68, 11.14,  
00466 9.663, 7.886, 6.505, 5.549, 4.931, 4.174, 3.014, 1.99, 1.339,  
00467 0.9012, 0.6096, 0.4231, 0.3152, 0.2701, 0.2561, 0.2696, 0.2523, 0.7171,  
00468 0.5333, 0.4876, 0.3218, 0.2536, 0.2178, 0.1861, 0.1546},  
00469 {5.229, 5.456, 6.192, 7.112, 8.094, 9.038, 9.776, 10.16, 9.992,  
00470 9, 7.493, 6.162, 5.154, 4.461, 3.788, 2.935, 2.058, 1.407,  
00471 0.9609, 0.6738, 0.4989, 0.3927, 0.3494, 0.3375, 0.3689, 0.3866, 0.6716,  
00472 0.7088, 0.6307, 0.4388, 0.3831, 0.3318, 0.2801, 0.2317},  
00473 {4.712, 4.75, 5.283, 6.062, 6.943, 7.874, 8.715, 9.344, 9.516,  
00474 8.913, 7.63, 6.223, 5.1, 4.346, 3.709, 2.982, 2.235, 1.605,  
00475 1.142, 0.8411, 0.6565, 0.5427, 0.4942, 0.4907, 0.5447, 0.6331, 0.9356,  
00476 0.7821, 0.7611, 0.663, 0.628, 0.5915, 0.5763, 0.5451},  
00477 {4.621, 4.6, 5.091, 5.827, 6.688, 7.624, 8.529, 9.276, 9.631,  
00478 9.219, 7.986, 6.499, 5.264, 4.401, 3.737, 2.996, 2.292, 1.69,  
00479 1.237, 0.9325, 0.7325, 0.6093, 0.5742, 0.5871, 0.6446, 0.6139, 0.9845,  
00480 0.9741, 1.044, 0.9091, 0.8661, 0.8218, 0.7617, 0.6884},  
00481 {4.647, 4.573, 5.038, 5.766, 6.61, 7.534, 8.489, 9.252, 9.6,  
00482 9.161, 7.958, 6.512, 5.259, 4.317, 3.547, 2.789, 2.13, 1.601,  
00483 1.205, 0.9321, 0.7532, 0.6464, 0.6173, 0.5896, 0.5782, 1.014, 1.096,  
00484 1.226, 1.387, 1.111, 1.042, 0.9908, 0.9408, 0.8311},  
00485 {4.621, 4.534, 4.984, 5.693, 6.545, 7.49, 8.444, 9.177, 9.531,  
00486 9.117, 7.928, 6.533, 5.27, 4.271, 3.431, 2.575, 1.902, 1.42,  
00487 1.11, 0.9004, 0.7658, 0.6955, 0.7676, 0.9088, 0.8989, 1.028, 1.221,  
00488 1.455, 1.583, 1.375, 1.376, 1.498, 1.744, 1.925},  
00489 {4.514, 4.41, 4.837, 5.545, 6.416, 7.38, 8.287, 9.05, 9.416,  
00490 9.022, 7.903, 6.496, 5.175, 4.111, 3.232, 2.38, 1.76, 1.335,  
00491 1.068, 0.9227, 0.8515, 0.8511, 0.9534, 1.091, 0.4909, 0.9377, 1.241,  
00492 1.592, 1.739, 1.478, 1.473, 1.597, 1.893, 2.117},  
00493 {4.407, 4.264, 4.61, 5.263, 6.095, 7.046, 8.005, 8.805, 9.201,  
00494 8.823, 7.705, 6.299, 4.964, 3.891, 3.046, 2.214, 1.641, 1.261,  
00495 1.027, 0.922, 0.8759, 0.8893, 0.9782, 0.3707, 0.4349, 0.976, 1.523,  
00496 2.021, 1.906, 1.524, 1.486, 1.53, 1.741, 1.869},  
00497 {4.156, 4.007, 4.37, 4.987, 5.777, 6.719, 7.728, 8.578, 8.97,  
00498 8.552, 7.409, 6.027, 4.731, 3.684, 2.814, 2.029, 1.501, 1.159,  
00499 0.9542, 0.8666, 0.8191, 0.8371, 0.9704, 0.3324, 0.5634, 0.9279, 1.512,  
00500 2.042, 1.951, 1.566, 1.535, 1.567, 1.739, 1.822},  
00501 {3.98, 3.883, 4.232, 4.841, 5.594, 6.502, 7.497, 8.296, 8.631,  
00502 8.161, 7.043, 5.703, 4.51, 3.548, 2.717, 1.951, 1.435, 1.107,  
00503 0.9188, 0.8286, 0.772, 0.7564, 0.8263, 0.3026, 0.3854, 0.8743, 1.452,  
00504 2.024, 2.012, 1.764, 1.85, 2.125, 2.621, 2.9},  
00505 {3.877, 3.811, 4.167, 4.733, 5.462, 6.324, 7.144, 7.862, 8.14,  
00506 7.695, 6.613, 5.393, 4.299, 3.42, 2.653, 1.947, 1.44, 1.1,  
00507 0.8953, 0.7907, 0.7218, 0.6881, 0.7281, 0.8846, 0.3802, 0.6853, 1.025,  
00508 1.593, 2.088, 1.846, 1.866, 2.108, 2.548, 2.737},  
00509 {3.636, 3.634, 4.034, 4.637, 5.354, 6.148, 6.899, 7.495, 7.657,  
00510 7.138, 6.107, 4.974, 3.983, 3.21, 2.558, 1.932, 1.439, 1.062,  
00511 0.8104, 0.6632, 0.5721, 0.517, 0.5182, 0.501, 0.3937, 0.5565, 0.7675,  
00512 0.9923, 1.455, 1.554, 1.567, 1.701, 2.077, 2.257},  
00513 {3.366, 3.446, 3.885, 4.473, 5.16, 5.9, 6.557, 6.973, 6.973,  
00514 6.379, 5.387, 4.385, 3.561, 2.941, 2.428, 1.848, 1.349, 0.9512,  
00515 0.6956, 0.5432, 0.4437, 0.3733, 0.3297, 0.2972, 0.2518, 0.4613, 0.5078,  
00516 0.6579, 0.8696, 0.9827, 1.028, 1.114, 1.275, 1.315},  
00517 {2.975, 3.121, 3.567, 4.146, 4.797, 5.466, 6.003, 6.279, 6.087,  
00518 5.372, 4.442, 3.614, 2.986, 2.521, 2.082, 1.532, 1.066, 0.7304,  
00519 0.5316, 0.4123, 0.3314, 0.274, 0.2365, 0.2061, 0.1755, 0.1348, 0.2217,  
00520 0.2854, 0.387, 0.4315, 0.4301, 0.4585, 0.5195, 0.5349},  
00521 {2.434, 2.632, 3.054, 3.577, 4.156, 4.725, 5.1, 5.109, 4.7,  
00522 3.958, 3.191, 2.588, 2.182, 1.866, 1.491, 1.034, 0.6936, 0.4774,  
00523 0.3551, 0.2749, 0.2203, 0.1829, 0.1544, 0.1325, 0.1103, 0.07716, 0.08626,  
00524 0.1037, 0.1455, 0.1418, 0.1351, 0.1339, 0.1407, 0.143},  
00525 {1.798, 2.004, 2.333, 2.746, 3.164, 3.49, 3.572, 3.315, 2.833,  
00526 2.269, 1.797, 1.485, 1.222, 0.9595, 0.6972, 0.4717, 0.3313, 0.2315,  
00527 0.1727, 0.1407, 0.1204, 0.1116, 0.08791, 0.07567, 0.07432, 0.04138,  
00528 0.02942,  
00529 0.02407, 0.02969, 0.02808, 0.0261, 0.02364, 0.02079, 0.01623},  
00530 {1.01, 1.1, 1.194, 1.284, 1.32, 1.278, 1.136, 0.9619, 0.815,  
00531 0.7226, 0.6136, 0.472, 0.3124, 0.2056, 0.1448, 0.1041, 0.07233, 0.05587,  
00532 0.05634, 0.04788, 0.03971, 0.02542, 0.01735, 0.01522, 0.01475, 0.01076,  
00533 0.005235,  
00534 0.003546, 0.003113, 0.003146, 0.002902, 0.002635, 0.001966, 0.001156},  
00535 {0.04181, 0.03949, 0.03577, 0.03191, 0.02839, 0.02391, 0.01796, 0.01251,  
00536 0.008422,  
00537 0.005359, 0.003257, 0.001769, 0.001092, 0.0008405, 0.001744, 0.0003061,  
00538 0.0002956, 0.000334,

```

00539      0.0004223, 0.0004062, 0.0003886, 0.0003418, 0.0003588, 0.0005709,
00540      0.001368, 0.0001472, 0.0003282,
00541      2.039e-06, 9.891e-13, 1.597e-13, 9.622e-14, 1.167e-13, 1.706e-13,
00542      3.59e-13}},
00543      {{3.361, 3.811, 4.35, 4.813, 5.132, 5.252, 5.019, 4.545, 3.913,
00544      3.25, 2.739, 2.417, 2.071, 1.582, 1.059, 0.6248, 0.3911, 0.2807,
00545      0.2095, 0.1539, 0.118, 0.0953, 0.08084, 0.07456, 0.07567, 0.07108, 0.1196,
00546      0.08475, 0.0839, 0.05724, 0.04673, 0.0412, 0.03626, 0.03134}},
00547      {3.193, 3.459, 3.973, 4.578, 5.172, 5.677, 5.945, 5.913, 5.487,
00548      4.686, 3.838, 3.217, 2.773, 2.409, 1.9, 1.33, 0.899, 0.64,
00549      0.4734, 0.355, 0.277, 0.2245, 0.1928, 0.1794, 0.1806, 0.1934, 0.3268,
00550      0.2217, 0.1969, 0.1375, 0.1206, 0.103, 0.08729, 0.07332}},
00551      {3.456, 3.617, 4.127, 4.778, 5.448, 6.083, 6.59, 6.858, 6.716,
00552      6.004, 4.975, 4.092, 3.442, 2.986, 2.474, 1.87, 1.328, 0.9231,
00553      0.6642, 0.5044, 0.4019, 0.336, 0.3049, 0.2996, 0.3227, 0.3571, 0.4295,
00554      0.3789, 0.3663, 0.3286, 0.3128, 0.3015, 0.302, 0.2968}},
00555      {3.743, 3.817, 4.311, 4.948, 5.666, 6.417, 7.077, 7.528, 7.551,
00556      6.893, 5.766, 4.721, 3.913, 3.326, 2.788, 2.144, 1.555, 1.093,
00557      0.7905, 0.6101, 0.4943, 0.4214, 0.3965, 0.3958, 0.4205, 0.4093, 0.5808,
00558      0.5876, 0.6359, 0.5956, 0.5731, 0.5485, 0.512, 0.4733}},
00559      {4.012, 4.012, 4.455, 5.113, 5.866, 6.661, 7.41, 7.98, 8.111,
00560      7.509, 6.364, 5.223, 4.308, 3.602, 2.951, 2.23, 1.625, 1.168,
00561      0.8748, 0.6916, 0.5758, 0.5122, 0.5053, 0.444, 0.4277, 0.7575, 0.7756,
00562      0.8848, 1.034, 0.8886, 0.8352, 0.7965, 0.7562, 0.68}},
00563      {4.136, 4.113, 4.549, 5.187, 5.93, 6.754, 7.592, 8.298, 8.546,
00564      8.022, 6.882, 5.668, 4.625, 3.778, 3.016, 2.236, 1.625, 1.193,
00565      0.9253, 0.756, 0.6523, 0.598, 0.6186, 0.6287, 0.4804, 0.8007, 0.9732,
00566      1.175, 1.329, 1.194, 1.193, 1.288, 1.47, 1.603}},
00567      {4.185, 4.119, 4.528, 5.186, 5.95, 6.801, 7.678, 8.404, 8.709,
00568      8.272, 7.179, 5.922, 4.767, 3.814, 2.996, 2.187, 1.599, 1.203,
00569      0.9596, 0.8307, 0.7617, 0.7489, 0.822, 0.9684, 0.4374, 0.7664, 1.044,
00570      1.36, 1.525, 1.331, 1.334, 1.447, 1.685, 1.862}},
00571      {4.213, 4.111, 4.507, 5.117, 5.882, 6.773, 7.67, 8.464, 8.829,
00572      8.438, 7.378, 6.088, 4.84, 3.802, 2.915, 2.114, 1.561, 1.195,
00573      0.9731, 0.8773, 0.8296, 0.8295, 0.9193, 1.114, 0.3986, 0.8793, 1.394,
00574      1.884, 1.786, 1.437, 1.403, 1.446, 1.64, 1.752}},
00575      {4.216, 4.092, 4.458, 5.04, 5.78, 6.67, 7.617, 8.47, 8.883,
00576      8.531, 7.504, 6.163, 4.862, 3.792, 2.891, 2.063, 1.519, 1.174,
00577      0.979, 0.8935, 0.8377, 0.8354, 0.9357, 0.3235, 0.4057, 0.9346, 1.53,
00578      2.078, 1.951, 1.555, 1.525, 1.557, 1.737, 1.825}},
00579      {4.168, 4.054, 4.417, 5.011, 5.76, 6.655, 7.601, 8.453, 8.882,
00580      8.539, 7.487, 6.143, 4.853, 3.809, 2.92, 2.088, 1.531, 1.181,
00581      0.9861, 0.8955, 0.8297, 0.8116, 0.8945, 1.137, 0.4003, 0.9318, 1.557,
00582      2.163, 2.095, 1.82, 1.931, 2.246, 2.823, 3.14}},
00583      {4.15, 4.078, 4.457, 5.044, 5.791, 6.692, 7.602, 8.343, 8.697,
00584      8.324, 7.282, 5.969, 4.743, 3.764, 2.926, 2.138, 1.572, 1.197,
00585      0.9808, 0.871, 0.7977, 0.764, 0.7663, 0.8665, 0.3969, 0.7472, 1.125,
00586      1.78, 2.277, 1.98, 2.036, 2.365, 2.936, 3.186}},
00587      {4.028, 4.006, 4.421, 5.039, 5.793, 6.657, 7.515, 8.216, 8.5,
00588      8.071, 7.01, 5.768, 4.638, 3.752, 3.021, 2.305, 1.714, 1.261,
00589      0.9646, 0.7989, 0.6935, 0.6314, 0.6426, 0.5914, 0.4424, 0.6827, 0.917,
00590      1.27, 1.864, 1.895, 1.922, 2.172, 2.772, 3.141}},
00591      {3.825, 3.864, 4.324, 4.97, 5.727, 6.565, 7.363, 7.978, 8.162,
00592      7.628, 6.542, 5.369, 4.372, 3.601, 2.977, 2.292, 1.697, 1.211,
00593      0.8808, 0.6852, 0.562, 0.4717, 0.4094, 0.366, 0.3093, 0.5639, 0.649,
00594      0.8282, 1.144, 1.27, 1.346, 1.522, 1.831, 1.939}},
00595      {3.493, 3.611, 4.102, 4.736, 5.491, 6.333, 7.098, 7.615, 7.666,
00596      7.006, 5.88, 4.791, 3.918, 3.283, 2.733, 2.099, 1.522, 1.066,
00597      0.7595, 0.5845, 0.4736, 0.3918, 0.337, 0.2962, 0.2564, 0.2024, 0.3657,
00598      0.4744, 0.6366, 0.76, 0.8059, 0.9015, 1.065, 1.126}},
00599      {3.167, 3.353, 3.857, 4.499, 5.225, 6.002, 6.699, 7.106, 6.98,
00600      6.192, 5.094, 4.125, 3.4, 2.87, 2.382, 1.774, 1.246, 0.8549,
00601      0.6089, 0.4666, 0.3762, 0.3082, 0.2643, 0.2283, 0.1942, 0.142, 0.1945,
00602      0.2512, 0.3593, 0.4048, 0.4081, 0.4206, 0.4616, 0.4738}},
00603      {2.839, 3.07, 3.565, 4.17, 4.846, 5.529, 6.041, 6.194, 5.833,
00604      5.009, 4.027, 3.239, 2.656, 2.227, 1.794, 1.282, 0.9003, 0.6219,
00605      0.4405, 0.3353, 0.2672, 0.2205, 0.1979, 0.1911, 0.1846, 0.1044, 0.09845,
00606      0.1069, 0.1547, 0.1611, 0.156, 0.1466, 0.1369, 0.1212}},
00607      {2.471, 2.75, 3.188, 3.712, 4.27, 4.77, 4.987, 4.802, 4.251,
00608      3.473, 2.731, 2.217, 1.83, 1.498, 1.137, 0.7594, 0.5383, 0.3859,
00609      0.2842, 0.2242, 0.1817, 0.1654, 0.1508, 0.1422, 0.1274, 0.07194, 0.05548,
00610      0.04058, 0.05226, 0.05347, 0.04942, 0.04415, 0.03177, 0.01923}},
00611      {2.192, 2.463, 2.735, 2.958, 3.106, 3.134, 2.927, 2.561, 2.149,
00612      1.825, 1.627, 1.429, 1.102, 0.766, 0.4943, 0.3166, 0.2184, 0.1471,
00613      0.1087, 0.08747, 0.07338, 0.06265, 0.05554, 0.05001, 0.04253, 0.03353,
00614      0.01781,
00615      0.01332, 0.01371, 0.01531, 0.01438, 0.01296, 0.00982, 0.006101}},
00616      {{0.2905, 0.282, 0.2639, 0.2366, 0.2078, 0.1879, 0.1644, 0.1407, 0.1163,
00617      0.08898, 0.06233, 0.04005, 0.02438, 0.01319, 0.007679, 0.005681, 0.004482,
00618      0.004329,
00619      0.004242, 0.004414, 0.004655, 0.004515, 0.003998, 0.00342, 0.003203,
00620      0.001672, 0.00135,
00621      0.001376, 0.0004656, 0.0003538, 0.000242, 0.0001647, 0.0001078,
00622      5.702e-05}},
00623      {1.5, 1.685, 1.901, 2.105, 2.216, 2.203, 2.048, 1.855, 1.609,
00624      1.357, 1.162, 1.012, 0.8042, 0.573, 0.372, 0.223, 0.144, 0.1095,
00625      0.08876, 0.07152, 0.05912, 0.05031, 0.04379, 0.03983, 0.03757, 0.03481,

```



```
00626 0.04463,
00627 0.02747, 0.02318, 0.01802, 0.0155, 0.01347, 0.01136, 0.009558},
00628 {2.228, 2.464, 2.889, 3.347, 3.773, 4.098, 4.191, 4.074, 3.7,
00629 3.132, 2.591, 2.217, 1.929, 1.594, 1.177, 0.7799, 0.5288, 0.3786,
00630 0.2805, 0.2144, 0.1695, 0.1403, 0.1236, 0.1157, 0.1134, 0.1148, 0.1127,
00631 0.1059, 0.0982, 0.08934, 0.08645, 0.08666, 0.09037, 0.09339},
00632 {2.761, 2.942, 3.406, 3.981, 4.592, 5.177, 5.591, 5.699, 5.391,
00633 4.682, 3.883, 3.267, 2.804, 2.378, 1.889, 1.347, 0.9255, 0.6382,
00634 0.4661, 0.361, 0.2904, 0.2459, 0.2237, 0.2143, 0.2142, 0.1959, 0.2522,
00635 0.2552, 0.2714, 0.274, 0.2676, 0.2624, 0.2589, 0.2436},
00636 {3.207, 3.317, 3.769, 4.373, 5.05, 5.759, 6.356, 6.677, 6.533,
00637 5.82, 4.861, 4.056, 3.414, 2.865, 2.322, 1.708, 1.204, 0.8386,
00638 0.6195, 0.4861, 0.3997, 0.3459, 0.32, 0.2881, 0.251, 0.4026, 0.4166,
00639 0.4798, 0.5723, 0.5711, 0.5498, 0.5348, 0.5181, 0.4851},
00640 {3.509, 3.549, 3.992, 4.607, 5.301, 6.054, 6.784, 7.302, 7.372,
00641 6.753, 5.706, 4.725, 3.909, 3.223, 2.58, 1.894, 1.356, 0.9716,
00642 0.7397, 0.5944, 0.4998, 0.4401, 0.4193, 0.3649, 0.3204, 0.5317, 0.5992,
00643 0.7541, 0.9257, 0.9193, 0.9167, 0.976, 1.081, 1.167},
00644 {3.748, 3.717, 4.128, 4.736, 5.472, 6.31, 7.086, 7.725, 7.909,
00645 7.387, 6.323, 5.206, 4.217, 3.376, 2.638, 1.912, 1.382, 1.026,
00646 0.8072, 0.6849, 0.6123, 0.5755, 0.6121, 0.6279, 0.3644, 0.5863, 0.823,
00647 1.068, 1.241, 1.134, 1.134, 1.222, 1.397, 1.55},
00648 {3.966, 3.863, 4.239, 4.826, 5.583, 6.465, 7.317, 8.033, 8.34,
00649 7.916, 6.843, 5.63, 4.49, 3.524, 2.698, 1.933, 1.409, 1.074,
00650 0.8747, 0.7756, 0.7211, 0.7103, 0.7513, 0.3147, 0.3627, 0.7582, 1.206,
00651 1.64, 1.639, 1.321, 1.285, 1.334, 1.496, 1.611},
00652 {4.184, 4.066, 4.42, 4.981, 5.712, 6.597, 7.502, 8.305, 8.719,
00653 8.382, 7.314, 6.02, 4.76, 3.702, 2.787, 1.993, 1.461, 1.13,
00654 0.9379, 0.8456, 0.7865, 0.7693, 0.835, 0.3132, 0.4006, 0.9192, 1.486,
00655 1.956, 1.836, 1.485, 1.462, 1.502, 1.65, 1.74},
00656 {4.304, 4.184, 4.566, 5.178, 5.931, 6.823, 7.761, 8.572, 9.013,
00657 8.72, 7.64, 6.283, 4.975, 3.89, 2.959, 2.123, 1.552, 1.194,
00658 0.9949, 0.8938, 0.8268, 0.814, 0.8837, 0.3629, 0.449, 1.027, 1.698,
00659 2.339, 2.232, 1.709, 1.64, 1.636, 1.794, 1.865},
00660 {4.354, 4.279, 4.704, 5.365, 6.153, 7.049, 7.975, 8.793, 9.218,
00661 8.889, 7.805, 6.42, 5.093, 4.013, 3.112, 2.283, 1.671, 1.263,
00662 1.032, 0.9124, 0.8364, 0.8112, 0.8709, 0.9784, 0.5043, 0.9917, 1.587,
00663 2.354, 2.467, 1.85, 1.738, 1.724, 1.873, 1.918},
00664 {4.35, 4.331, 4.785, 5.473, 6.289, 7.199, 8.133, 8.898, 9.288,
00665 8.907, 7.796, 6.411, 5.112, 4.065, 3.212, 2.421, 1.788, 1.331,
00666 1.04, 0.8749, 0.769, 0.7242, 0.7799, 0.8189, 0.5724, 0.86, 1.151,
00667 1.587, 1.946, 1.722, 1.71, 1.845, 2.146, 2.307},
00668 {4.268, 4.264, 4.774, 5.45, 6.298, 7.255, 8.148, 8.907, 9.242,
00669 8.793, 7.612, 6.244, 5.029, 4.124, 3.411, 2.65, 1.98, 1.442,
00670 1.074, 0.8493, 0.7077, 0.6157, 0.5826, 0.5377, 0.4848, 0.7245, 0.8927,
00671 1.153, 1.565, 1.551, 1.638, 1.846, 2.153, 2.304},
00672 {4.119, 4.16, 4.652, 5.372, 6.224, 7.158, 8.06, 8.773, 9.034,
00673 8.507, 7.307, 5.989, 4.866, 4.046, 3.387, 2.637, 1.959, 1.402,
00674 1.01, 0.7722, 0.6266, 0.527, 0.462, 0.4124, 0.3452, 0.6418, 0.572,
00675 0.7352, 1.009, 1.245, 1.364, 1.556, 1.828, 1.914},
00676 {3.932, 4.041, 4.572, 5.285, 6.144, 7.099, 7.985, 8.671, 8.849,
00677 8.203, 6.936, 5.657, 4.621, 3.877, 3.266, 2.526, 1.845, 1.302,
00678 0.9214, 0.6906, 0.5486, 0.4553, 0.4002, 0.3533, 0.3033, 0.24, 0.4062,
00679 0.5084, 0.7133, 0.9299, 0.9714, 1.073, 1.204, 1.211},
00680 {3.882, 4.074, 4.658, 5.414, 6.269, 7.182, 8.05, 8.635, 8.639,
00681 7.828, 6.529, 5.303, 4.339, 3.626, 3.001, 2.262, 1.617, 1.128,
00682 0.7869, 0.574, 0.4488, 0.3713, 0.3216, 0.2794, 0.2403, 0.1962, 0.2638,
00683 0.3018, 0.4483, 0.5618, 0.5717, 0.5812, 0.5993, 0.5956},
00684 {4.195, 4.559, 5.287, 6.13, 7.024, 7.911, 8.623, 8.913, 8.545,
00685 7.44, 6.059, 4.92, 4.063, 3.391, 2.747, 1.953, 1.327, 0.9036,
00686 0.6235, 0.4566, 0.3534, 0.2935, 0.2547, 0.2226, 0.1871, 0.1494, 0.1959,
00687 0.1797, 0.2736, 0.3214, 0.3054, 0.2803, 0.2443, 0.1504},
00688 {4.644, 5.211, 6.192, 7.373, 8.534, 9.582, 10.35, 10.37, 9.49,
00689 7.907, 6.299, 5.088, 4.224, 3.492, 2.701, 1.795, 1.184, 0.8001,
00690 0.5348, 0.3789, 0.2861, 0.2435, 0.2055, 0.1754, 0.1605, 0.1076, 0.1644,
00691 0.09069, 0.1624, 0.2356, 0.223, 0.2001, 0.164, 0.09807},
00692 {{0.0001362, 0.0001753, 0.0001792, 0.0001881, 0.0002233, 0.0002056,
00693 0.0001083, 1.505e-05, 1.073e-05,
00694 1.22e-05, 7.475e-06, 3.518e-06, 1.292e-06, 3.812e-07, 8.611e-08,
00695 3.148e-09, 1.29e-09, 2.111e-08,
00696 7.591e-07, 1.125e-06, 4.905e-06, 1.473e-05, 3.597e-05, 9.35e-05,
00697 0.0002623, 0.0001066, 0.0002064,
00698 0.0001924, 6.532e-10, 3.205e-10, 3.838e-10, 3.541e-10, 3.331e-10,
00699 2.774e-10},
00700 {0.1865, 0.1757, 0.1596, 0.1454, 0.1343, 0.125, 0.1033, 0.08324, 0.06202,
00701 0.04341, 0.0295, 0.0187, 0.01129, 0.007115, 0.004932, 0.003824, 0.002733,
00702 0.003019,
00703 0.004022, 0.004428, 0.004414, 0.003826, 0.003009, 0.002117, 0.001375,
00704 0.0009066, 0.001031,
00705 0.0004673, 0.0002194, 0.0001683, 0.0001154, 6.607e-05, 3.128e-05,
00706 1.601e-05},
00707 {1.238, 1.384, 1.576, 1.788, 1.95, 1.993, 1.845, 1.593, 1.329,
00708 1.098, 0.9387, 0.8174, 0.6531, 0.4579, 0.2899, 0.1818, 0.1253, 0.09461,
00709 0.0741, 0.05946, 0.04989, 0.04353, 0.03862, 0.03469, 0.03181, 0.02862,
00710 0.02499,
00711 0.02062, 0.0178, 0.01681, 0.0162, 0.01679, 0.01921, 0.02192},
00712 {1.92, 2.106, 2.478, 2.931, 3.421, 3.86, 4.043, 3.866, 3.41,
```

```

00713      2.84, 2.372, 2.077, 1.825, 1.472, 1.07, 0.7023, 0.4688, 0.3296,
00714      0.2453, 0.1889, 0.15, 0.1246, 0.1094, 0.09788, 0.09079, 0.08267, 0.08977,
00715      0.09461, 0.09867, 0.09977, 0.09675, 0.09757, 0.1011, 0.1004},
00716      {2.505, 2.661, 3.095, 3.659, 4.295, 4.928, 5.368, 5.413, 5.018,
00717      4.291, 3.582, 3.052, 2.636, 2.193, 1.714, 1.199, 0.8121, 0.5616,
00718      0.4193, 0.3284, 0.2656, 0.2245, 0.2009, 0.1819, 0.1615, 0.2142, 0.2142,
00719      0.2523, 0.3029, 0.3253, 0.3248, 0.323, 0.3186, 0.3005},
00720      {3.053, 3.153, 3.582, 4.16, 4.823, 5.525, 6.111, 6.38, 6.176,
00721      5.453, 4.575, 3.84, 3.219, 2.652, 2.09, 1.509, 1.055, 0.7478,
00722      0.5678, 0.4529, 0.3746, 0.322, 0.2979, 0.2602, 0.284, 0.3515, 0.39,
00723      0.4915, 0.6421, 0.6745, 0.684, 0.7208, 0.7815, 0.8179},
00724      {3.422, 3.44, 3.875, 4.477, 5.191, 5.965, 6.661, 7.123, 7.132,
00725      6.48, 5.463, 4.526, 3.718, 3.005, 2.344, 1.688, 1.206, 0.8837,
00726      0.6856, 0.5649, 0.4863, 0.445, 0.4856, 0.5006, 0.3136, 0.4515, 0.6109,
00727      0.7863, 0.9684, 0.937, 0.937, 0.9949, 1.106, 1.204},
00728      {3.782, 3.708, 4.085, 4.684, 5.42, 6.264, 7.105, 7.757, 7.965,
00729      7.421, 6.309, 5.169, 4.161, 3.312, 2.538, 1.814, 1.314, 0.9921,
00730      0.7986, 0.6906, 0.6241, 0.6037, 0.6883, 0.3135, 0.3539, 0.6781, 1.044,
00731      1.41, 1.465, 1.198, 1.162, 1.195, 1.315, 1.399},
00732      {4.093, 3.956, 4.303, 4.881, 5.64, 6.546, 7.473, 8.261, 8.617,
00733      8.211, 7.109, 5.806, 4.611, 3.616, 2.718, 1.923, 1.396, 1.077,
00734      0.8999, 0.7993, 0.7318, 0.69, 0.7736, 0.3307, 0.5444, 0.8973, 1.424,
00735      1.867, 1.76, 1.423, 1.397, 1.429, 1.558, 1.631},
00736      {4.316, 4.179, 4.572, 5.214, 6.033, 6.982, 7.93, 8.754, 9.154,
00737      8.809, 7.719, 6.283, 4.962, 3.898, 2.961, 2.106, 1.526, 1.178,
00738      0.9911, 0.8826, 0.8091, 0.792, 0.8984, 0.4081, 0.6591, 1.079, 1.762,
00739      2.394, 2.262, 1.722, 1.649, 1.64, 1.793, 1.856},
00740      {4.422, 4.322, 4.787, 5.478, 6.345, 7.323, 8.254, 9.118, 9.564,
00741      9.218, 8.075, 6.604, 5.203, 4.058, 3.117, 2.274, 1.677, 1.284,
00742      1.048, 0.9151, 0.8297, 0.7966, 0.8935, 0.5312, 0.7531, 1.072, 1.715,
00743      2.495, 2.546, 1.897, 1.782, 1.77, 1.93, 1.983},
00744      {4.547, 4.469, 4.942, 5.688, 6.549, 7.488, 8.486, 9.332, 9.775,
00745      9.417, 8.264, 6.785, 5.379, 4.249, 3.33, 2.505, 1.852, 1.39,
00746      1.098, 0.9308, 0.8263, 0.7895, 0.8576, 1.037, 0.635, 0.9803, 1.371,
00747      1.867, 2.133, 1.823, 1.82, 1.997, 2.372, 2.584},
00748      {4.687, 4.634, 5.104, 5.806, 6.687, 7.693, 8.664, 9.479, 9.89,
00749      9.543, 8.412, 6.918, 5.518, 4.426, 3.541, 2.711, 2.014, 1.493,
00750      1.144, 0.9317, 0.797, 0.7099, 0.6575, 0.6067, 0.526, 0.9154, 1.118,
00751      1.507, 1.832, 1.788, 1.887, 2.179, 2.647, 2.868},
00752      {4.695, 4.698, 5.185, 5.955, 6.871, 7.893, 8.877, 9.693, 10.1,
00753      9.723, 8.528, 6.981, 5.591, 4.567, 3.788, 2.967, 2.242, 1.655,
00754      1.222, 0.9406, 0.7599, 0.6428, 0.57, 0.513, 0.4496, 0.9249, 0.8597,
00755      1.064, 1.428, 1.661, 1.831, 2.151, 2.551, 2.772},
00756      {4.817, 4.823, 5.364, 6.154, 7.109, 8.194, 9.225, 10.09, 10.53,
00757      10.13, 8.82, 7.204, 5.79, 4.774, 3.99, 3.147, 2.374, 1.734,
00758      1.249, 0.9272, 0.7213, 0.5919, 0.522, 0.4691, 0.4205, 0.3435, 0.704,
00759      0.795, 1.078, 1.369, 1.506, 1.697, 1.921, 2.057},
00760      {5.178, 5.275, 5.904, 6.794, 7.806, 8.901, 9.941, 10.76, 11.06,
00761      10.41, 8.899, 7.194, 5.78, 4.767, 3.992, 3.132, 2.304, 1.622,
00762      1.125, 0.8041, 0.6135, 0.4964, 0.4334, 0.388, 0.3494, 0.3124, 0.5608,
00763      0.6202, 0.8549, 1.142, 1.185, 1.327, 1.469, 1.521},
00764      {5.765, 5.982, 6.854, 7.999, 9.258, 10.55, 11.75, 12.58, 12.65,
00765      11.51, 9.567, 7.638, 6.122, 5.016, 4.17, 3.208, 2.283, 1.528,
00766      0.9814, 0.6677, 0.4925, 0.3884, 0.3284, 0.2876, 0.2571, 0.2207, 0.5107,
00767      0.5167, 0.7005, 0.8149, 0.7878, 0.7521, 0.6899, 0.4697},
00768      {5.862, 6.014, 6.876, 8.048, 9.368, 10.8, 12.34, 13.65, 13.83,
00769      12.41, 10.19, 8.106, 6.468, 5.285, 4.422, 3.392, 2.324, 1.509,
00770      0.9796, 0.6834, 0.5019, 0.387, 0.3295, 0.2804, 0.2425, 0.2084, 0.4694,
00771      0.4497, 0.5949, 0.6504, 0.6155, 0.5696, 0.5077, 0.2744}},
00772      {{6.617e-05, 8.467e-05, 8.509e-05, 9.824e-05, 0.0001317, 0.0001499,
00773      0.0001104, 1.226e-05, 1.003e-05,
00774      1.345e-05, 8.036e-06, 2.28e-06, 3.166e-07, 1.803e-08, 6.079e-10,
00775      3.031e-10, 1.336e-09, 1.748e-08,
00776      3.057e-07, 8.184e-07, 1.95e-06, 2.238e-06, 4.658e-06, 1.393e-05,
00777      4.326e-05, 3.288e-05, 0.0001662,
00778      8.012e-05, 6.48e-10, 3.371e-10, 4.509e-10, 4.052e-10, 3.677e-10,
00779      2.996e-10},
00780      {0.003936, 0.002158, 0.001688, 0.001555, 0.001396, 0.0008637, 0.0003091,
00781      4.908e-05, 1.173e-05,
00782      1.326e-05, 9.097e-06, 3.324e-06, 6.336e-07, 8.553e-08, 4.851e-09,
00783      3.058e-09, 9.169e-09, 2.451e-07,
00784      8.77e-07, 2.973e-07, 2.556e-07, 2.79e-07, 8.674e-07, 4.17e-06, 2.373e-05,
00785      3.876e-05, 6.493e-05,
00786      2.13e-05, 2.574e-09, 2.676e-10, 2.356e-10, 1.947e-10, 2.376e-10,
00787      1.955e-10},
00788      {0.8642, 0.927, 0.9782, 1.001, 0.9705, 0.885, 0.7521, 0.6223, 0.5357,
00789      0.4614, 0.373, 0.2955, 0.2138, 0.1342, 0.07775, 0.04773, 0.0348, 0.02871,
00790      0.02503, 0.02329, 0.02308, 0.02227, 0.01967, 0.01687, 0.01381, 0.008063,
00791      0.006489,
00792      0.005433, 0.004804, 0.00487, 0.004735, 0.004696, 0.004587, 0.003634},
00793      {1.584, 1.731, 2.006, 2.368, 2.742, 3.047, 3.124, 2.894, 2.517,
00794      2.087, 1.761, 1.588, 1.399, 1.091, 0.7549, 0.4844, 0.3282, 0.2357,
00795      0.1737, 0.1299, 0.1001, 0.08162, 0.0699, 0.06056, 0.05333, 0.07014,
00796      0.04552,
00797      0.04907, 0.04781, 0.04938, 0.04864, 0.04971, 0.05141, 0.05078},
00798      {2.256, 2.4, 2.786, 3.287, 3.854, 4.41, 4.773, 4.776, 4.396,
00799      3.755, 3.16, 2.747, 2.381, 1.94, 1.458, 0.9865, 0.6576, 0.4582,

```

00800 0.3447, 0.2672, 0.2121, 0.1768, 0.1538, 0.1358, 0.1219, 0.1466, 0.1371,  
00801 0.161, 0.1896, 0.2116, 0.2133, 0.2153, 0.2145, 0.2042},  
00802 {2.844, 2.913, 3.296, 3.841, 4.473, 5.127, 5.672, 5.933, 5.724,  
00803 5.005, 4.187, 3.549, 3.006, 2.486, 1.928, 1.368, 0.9363, 0.6524,  
00804 0.4946, 0.3921, 0.3189, 0.2696, 0.2503, 0.2181, 0.2227, 0.2946, 0.297,  
00805 0.3704, 0.4834, 0.5473, 0.5565, 0.581, 0.6189, 0.6392},  
00806 {3.309, 3.337, 3.765, 4.35, 5.025, 5.756, 6.438, 6.908, 6.947,  
00807 6.3, 5.252, 4.33, 3.576, 2.912, 2.258, 1.618, 1.15, 0.8271,  
00808 0.6351, 0.5129, 0.4287, 0.3846, 0.4351, 0.4889, 0.3229, 0.4124, 0.4839,  
00809 0.6285, 0.8468, 0.9091, 0.9059, 0.9413, 1.007, 1.074},  
00810 {3.725, 3.662, 4.042, 4.647, 5.381, 6.204, 7.019, 7.636, 7.841,  
00811 7.318, 6.177, 5.04, 4.09, 3.299, 2.545, 1.822, 1.303, 0.967,  
00812 0.7683, 0.6506, 0.5816, 0.5712, 0.667, 0.3774, 0.3833, 0.5497, 0.7343,  
00813 0.9899, 1.25, 1.213, 1.147, 1.144, 1.27, 1.389},  
00814 {4.074, 3.916, 4.273, 4.851, 5.624, 6.564, 7.525, 8.296, 8.632,  
00815 8.184, 7.013, 5.711, 4.593, 3.691, 2.813, 2.003, 1.445, 1.097,  
00816 0.8982, 0.7815, 0.7103, 0.6929, 0.7841, 0.3697, 0.42, 0.8027, 1.255,  
00817 1.716, 1.727, 1.411, 1.379, 1.416, 1.548, 1.618},  
00818 {4.378, 4.233, 4.657, 5.297, 6.121, 7.098, 8.11, 8.937, 9.31,  
00819 8.877, 7.696, 6.289, 5.029, 3.997, 3.066, 2.189, 1.589, 1.209,  
00820 0.9953, 0.8642, 0.7883, 0.7739, 0.9011, 0.4115, 0.6312, 0.9904, 1.555,  
00821 2.033, 1.915, 1.547, 1.523, 1.55, 1.675, 1.75},  
00822 {4.574, 4.464, 4.871, 5.569, 6.471, 7.487, 8.486, 9.356, 9.788,  
00823 9.387, 8.196, 6.717, 5.357, 4.245, 3.29, 2.387, 1.738, 1.316,  
00824 1.064, 0.9101, 0.8224, 0.8068, 0.9022, 0.478, 0.7124, 1.093, 1.75,  
00825 2.355, 2.233, 1.721, 1.663, 1.652, 1.836, 1.906},  
00826 {4.771, 4.625, 5.041, 5.796, 6.698, 7.684, 8.72, 9.611, 10.08,  
00827 9.736, 8.573, 7.013, 5.571, 4.447, 3.501, 2.606, 1.915, 1.439,  
00828 1.139, 0.956, 0.8517, 0.8306, 0.9424, 1.177, 0.5869, 1.036, 1.44,  
00829 1.878, 2.039, 1.705, 1.737, 1.923, 2.385, 2.641},  
00830 {5.006, 4.866, 5.324, 6.09, 7.006, 8.036, 9.084, 9.893, 10.33,  
00831 10.02, 8.839, 7.228, 5.736, 4.611, 3.71, 2.8, 2.063, 1.532,  
00832 1.183, 0.9673, 0.8363, 0.7853, 0.8359, 0.8923, 0.569, 1.039, 1.365,  
00833 1.742, 1.904, 1.758, 1.875, 2.251, 2.843, 3.224},  
00834 {5.099, 4.961, 5.447, 6.211, 7.152, 8.22, 9.322, 10.2, 10.72,  
00835 10.48, 9.248, 7.559, 5.993, 4.842, 3.975, 3.116, 2.35, 1.736,  
00836 1.297, 1.003, 0.8134, 0.702, 0.6582, 0.6213, 0.5842, 0.5417, 1.158,  
00837 1.4, 1.637, 1.863, 2.063, 2.497, 3.122, 3.523},  
00838 {5.329, 5.217, 5.717, 6.57, 7.585, 8.724, 9.852, 10.78, 11.29,  
00839 10.97, 9.671, 7.914, 6.3, 5.116, 4.263, 3.389, 2.58, 1.892,  
00840 1.383, 1.036, 0.8131, 0.6799, 0.6149, 0.5698, 0.5283, 0.4671, 1.093,  
00841 1.188, 1.482, 1.733, 1.908, 2.209, 2.612, 2.814},  
00842 {6.071, 6.113, 6.75, 7.703, 8.826, 10.05, 11.16, 11.96, 12.25,  
00843 11.61, 10.02, 8.102, 6.438, 5.24, 4.366, 3.448, 2.553, 1.795,  
00844 1.245, 0.8829, 0.6737, 0.5561, 0.4996, 0.4579, 0.4257, 0.4016, 0.968,  
00845 1.042, 1.284, 1.504, 1.639, 1.849, 2.098, 2.255},  
00846 {6.417, 6.432, 7.23, 8.422, 9.846, 11.4, 12.82, 13.78, 14,  
00847 12.98, 10.94, 8.756, 6.951, 5.69, 4.77, 3.716, 2.672, 1.818,  
00848 1.224, 0.8517, 0.6496, 0.5355, 0.4766, 0.4254, 0.389, 0.351, 0.9729,  
00849 0.9755, 1.108, 1.026, 0.9967, 0.9687, 0.9329, 0.8183},  
00850 {6.462, 6.445, 7.221, 8.396, 9.796, 11.38, 12.96, 14.19, 14.59,  
00851 13.52, 11.33, 9.046, 7.189, 5.92, 5.015, 3.949, 2.77, 1.841,  
00852 1.239, 0.86, 0.6441, 0.5179, 0.446, 0.42, 0.3818, 0.3068, 0.9267,  
00853 0.8849, 0.9828, 0.8458, 0.7544, 0.6963, 0.6233, 0.5358},  
00854 {{3.24e-05, 4.086e-05, 4.221e-05, 5.381e-05, 8.808e-05, 0.0001261,  
00855 0.0001263, 2.932e-05, 2.057e-05,  
00856 4.321e-05, 3.43e-05, 1.213e-05, 1.862e-06, 1.067e-07, 2.983e-09,  
00857 2.578e-08, 2.254e-09, 1.504e-08,  
00858 1.599e-07, 1.523e-07, 2.033e-07, 5.271e-07, 1.417e-06, 4.518e-06,  
00859 1.358e-05, 9.266e-06, 0.0001569,  
00860 3.355e-05, 7.653e-10, 3.889e-10, 5.082e-10, 4.498e-10, 4.072e-10,  
00861 3.302e-10},  
00862 {0.0688, 0.05469, 0.04502, 0.03604, 0.02861, 0.02128, 0.01453, 0.00924,  
00863 0.00574,  
00864 0.003447, 0.001888, 0.001147, 0.001154, 0.001548, 0.001193, 6.955e-05,  
00865 0.0003258, 1.673e-05,  
00866 1.393e-05, 1.849e-05, 2.841e-05, 4.045e-05, 5.864e-05, 4.426e-05,  
00867 1.722e-05, 3.186e-05, 5.582e-05,  
00868 6.823e-06, 3.559e-09, 3.282e-10, 2.847e-10, 2.3e-10, 2.728e-10,  
00869 2.237e-10},  
00870 {1.056, 1.158, 1.28, 1.377, 1.429, 1.395, 1.242, 1.043, 0.8709,  
00871 0.7435, 0.6183, 0.4824, 0.3665, 0.2668, 0.1749, 0.098, 0.09722, 0.08524,  
00872 0.05545, 0.04501, 0.04476, 0.04491, 0.04231, 0.03868, 0.03336, 0.01108,  
00873 0.01596,  
00874 0.009172, 0.008499, 0.008763, 0.008672, 0.008872, 0.008926, 0.007552},  
00875 {1.798, 1.952, 2.239, 2.589, 2.958, 3.261, 3.328, 3.12, 2.741,  
00876 2.293, 1.917, 1.686, 1.53, 1.28, 0.9362, 0.6052, 0.4093, 0.294,  
00877 0.2166, 0.1609, 0.1206, 0.09532, 0.08094, 0.07007, 0.06252, 0.1111,  
00878 0.04996,  
00879 0.0572, 0.0583, 0.06137, 0.06077, 0.0616, 0.0644, 0.06454},  
00880 {2.42, 2.566, 2.952, 3.45, 3.987, 4.499, 4.853, 4.878, 4.55,  
00881 3.918, 3.264, 2.802, 2.467, 2.094, 1.632, 1.127, 0.7523, 0.5168,  
00882 0.386, 0.2982, 0.2346, 0.1928, 0.1694, 0.1534, 0.1384, 0.1672, 0.1434,  
00883 0.1726, 0.2115, 0.2414, 0.2445, 0.2487, 0.2496, 0.2381},  
00884 {3.026, 3.096, 3.501, 4.016, 4.608, 5.244, 5.786, 6.088, 5.975,  
00885 5.326, 4.429, 3.701, 3.142, 2.636, 2.071, 1.477, 1.015, 0.7056,  
00886 0.5315, 0.4215, 0.3413, 0.2869, 0.2681, 0.2507, 0.2616, 0.3004, 0.3042,

```
00887    0.3879, 0.5203, 0.6045, 0.6196, 0.651, 0.6986, 0.7171},
00888    {3.463, 3.475, 3.878, 4.441, 5.112, 5.838, 6.522, 7.015, 7.096,
00889     6.52, 5.487, 4.496, 3.734, 3.105, 2.456, 1.776, 1.26, 0.9016,
00890     0.6841, 0.5485, 0.454, 0.404, 0.4609, 0.5432, 0.376, 0.4702, 0.5045,
00891     0.6555, 0.9157, 1.076, 1.113, 1.216, 1.372, 1.497},
00892    {3.745, 3.679, 4.071, 4.666, 5.394, 6.22, 7.064, 7.725, 7.961,
00893     7.448, 6.321, 5.161, 4.233, 3.472, 2.73, 1.96, 1.409, 1.04,
00894     0.8213, 0.6863, 0.6068, 0.5872, 0.7122, 0.9507, 0.4302, 0.5845, 0.768,
00895     1.036, 1.308, 1.372, 1.393, 1.57, 2.023, 2.387},
00896    {4.046, 3.927, 4.286, 4.869, 5.645, 6.599, 7.567, 8.4, 8.74,
00897     8.247, 7.036, 5.735, 4.638, 3.741, 2.905, 2.066, 1.493, 1.132,
00898     0.9307, 0.7993, 0.7306, 0.7334, 0.8811, 1.197, 0.4369, 0.7932, 1.239,
00899     1.716, 1.764, 1.639, 1.738, 2.042, 2.543, 2.814},
00900    {4.303, 4.169, 4.566, 5.224, 6.055, 7.053, 8.124, 9.037, 9.471,
00901     8.998, 7.714, 6.277, 4.981, 3.935, 3.018, 2.182, 1.599, 1.224,
00902     1.01, 0.8634, 0.7916, 0.7955, 0.9234, 1.237, 0.6362, 0.9588, 1.513,
00903     1.996, 1.902, 1.545, 1.53, 1.572, 1.712, 1.768},
00904    {4.506, 4.381, 4.819, 5.529, 6.428, 7.462, 8.518, 9.453, 9.907,
00905     9.467, 8.237, 6.703, 5.29, 4.149, 3.213, 2.346, 1.732, 1.324,
00906     1.075, 0.9181, 0.8397, 0.8375, 0.9434, 0.5018, 0.5414, 1.083, 1.74,
00907     2.347, 2.227, 1.721, 1.661, 1.648, 1.831, 1.912},
00908    {4.687, 4.558, 4.978, 5.725, 6.658, 7.731, 8.789, 9.675, 10.13,
00909     9.748, 8.526, 6.958, 5.498, 4.351, 3.431, 2.537, 1.87, 1.415,
00910     1.127, 0.9521, 0.8587, 0.8423, 0.8885, 1.184, 0.5938, 1.064, 1.469,
00911     1.877, 2.006, 1.669, 1.7, 1.879, 2.347, 2.626},
00912    {4.936, 4.809, 5.237, 6.008, 6.937, 7.97, 9.016, 9.924, 10.41,
00913     10.04, 8.758, 7.153, 5.667, 4.529, 3.633, 2.737, 2.011, 1.493,
00914     1.161, 0.9601, 0.8439, 0.8078, 0.8976, 1.023, 0.6551, 1.239, 1.595,
00915     1.9, 1.923, 1.721, 1.833, 2.227, 2.857, 3.283},
00916    {5.075, 4.962, 5.428, 6.241, 7.2, 8.253, 9.328, 10.2, 10.65,
00917     10.28, 8.983, 7.308, 5.772, 4.63, 3.752, 2.906, 2.151, 1.57,
00918     1.188, 0.9525, 0.8028, 0.7168, 0.7021, 0.7201, 0.7339, 0.6036, 1.423,
00919     1.574, 1.715, 1.841, 2.05, 2.525, 3.214, 3.663},
00920    {5.33, 5.234, 5.731, 6.529, 7.518, 8.638, 9.699, 10.59, 11.06,
00921     10.69, 9.363, 7.602, 6.028, 4.875, 4.029, 3.2, 2.426, 1.77,
00922     1.305, 0.9937, 0.7977, 0.6831, 0.6247, 0.6058, 0.5859, 0.5171, 1.306,
00923     1.309, 1.552, 1.652, 1.83, 2.179, 2.619, 2.844},
00924    {5.866, 5.848, 6.455, 7.346, 8.396, 9.533, 10.53, 11.34, 11.68,
00925     11.13, 9.619, 7.806, 6.227, 5.073, 4.235, 3.4, 2.554, 1.805,
00926     1.264, 0.9221, 0.7291, 0.623, 0.5765, 0.5499, 0.5313, 0.5058, 1.13,
00927     1.193, 1.384, 1.521, 1.658, 1.903, 2.204, 2.394},
00928    {6.303, 6.398, 7.222, 8.351, 9.705, 11.15, 12.37, 13.21, 13.37,
00929     12.4, 10.47, 8.393, 6.679, 5.472, 4.597, 3.653, 2.67, 1.827,
00930     1.234, 0.8789, 0.697, 0.5977, 0.5427, 0.5065, 0.4765, 0.4416, 1.108,
00931     1.122, 1.2, 1.026, 0.9995, 0.9762, 0.9406, 0.8155},
00932    {6.374, 6.361, 7.22, 8.387, 9.771, 11.27, 12.69, 13.72, 14.07,
00933     13.07, 10.88, 8.681, 6.947, 5.79, 4.913, 3.885, 2.773, 1.855,
00934     1.244, 0.9142, 0.7256, 0.598, 0.5151, 0.5035, 0.4732, 0.3585, 1.059,
00935     1.045, 1.089, 0.8483, 0.7697, 0.7077, 0.6288, 0.5436},
00936    {{0.03789, 0.03408, 0.03134, 0.02846, 0.02395, 0.01876, 0.01296, 0.008683,
00937     0.005595,
00938     0.003327, 0.001899, 0.001016, 0.0006645, 0.001147, 0.000686, 0.0006987,
00939     0.0006744, 0.0003903,
00940     1.823e-05, 1.418e-05, 1.097e-05, 9.197e-06, 8.385e-06, 5.361e-06,
00941     1.047e-05, 8.724e-06, 0.0001224,
00942     1.608e-05, 7.761e-10, 4.005e-10, 5.216e-10, 4.653e-10, 4.282e-10,
00943     3.472e-10},
00944    {0.9685, 1.035, 1.103, 1.161, 1.165, 1.106, 0.9814, 0.8435, 0.7222,
00945     0.6291, 0.5477, 0.4518, 0.3058, 0.2105, 0.1492, 0.08369, 0.1841, 0.1379,
00946     0.1032, 0.06085, 0.03194, 0.02288, 0.0206, 0.02385, 0.0298, 0.0102,
00947     0.002722,
00948     0.00743, 0.007294, 0.006871, 0.006275, 0.005435, 0.004153, 0.002587},
00949    {1.842, 2.002, 2.295, 2.618, 2.95, 3.205, 3.226, 2.966, 2.55,
00950     2.078, 1.668, 1.355, 1.079, 0.8142, 0.7131, 0.5158, 0.3251, 0.2412,
00951     0.1924, 0.1651, 0.1599, 0.164, 0.1325, 0.1215, 0.09591, 0.1117, 0.03993,
00952     0.04814, 0.04778, 0.04729, 0.04627, 0.0467, 0.04597, 0.04172},
00953    {2.489, 2.643, 3.032, 3.507, 4.035, 4.533, 4.829, 4.772, 4.375,
00954     3.714, 3.06, 2.551, 2.207, 1.938, 1.628, 1.185, 0.8109, 0.5625,
00955     0.4101, 0.3078, 0.2325, 0.1813, 0.1507, 0.1311, 0.1178, 0.1964, 0.1063,
00956     0.1261, 0.1425, 0.1561, 0.1546, 0.1532, 0.1543, 0.1498},
00957    {3.03, 3.156, 3.579, 4.144, 4.775, 5.42, 5.935, 6.129, 5.878,
00958     5.138, 4.235, 3.481, 2.956, 2.542, 2.086, 1.502, 1.028, 0.7057,
00959     0.5223, 0.4097, 0.3291, 0.2729, 0.2417, 0.2235, 0.2035, 0.2565, 0.2228,
00960     0.274, 0.339, 0.387, 0.3918, 0.396, 0.3931, 0.3692},
00961    {3.447, 3.508, 3.934, 4.533, 5.226, 5.96, 6.611, 7.003, 6.968,
00962     6.331, 5.301, 4.324, 3.576, 3.009, 2.45, 1.808, 1.285, 0.894,
00963     0.6612, 0.5244, 0.4273, 0.3588, 0.3341, 0.3096, 0.296, 0.4036, 0.403,
00964     0.5114, 0.6763, 0.7735, 0.7969, 0.8461, 0.9229, 0.9621},
00965    {3.722, 3.706, 4.103, 4.704, 5.429, 6.225, 6.997, 7.531, 7.688,
00966     7.197, 6.151, 5.021, 4.119, 3.436, 2.786, 2.084, 1.513, 1.082,
00967     0.808, 0.6428, 0.5308, 0.473, 0.5475, 0.6436, 0.4784, 0.5735, 0.5953,
00968     0.7959, 1.106, 1.287, 1.339, 1.478, 1.689, 1.86},
00969    {3.921, 3.841, 4.189, 4.783, 5.529, 6.4, 7.263, 7.978, 8.293,
00970     7.875, 6.788, 5.549, 4.498, 3.668, 2.902, 2.117, 1.551, 1.158,
00971     0.9124, 0.7546, 0.6628, 0.6526, 0.7867, 0.9666, 0.4912, 0.664, 0.8688,
00972     1.172, 1.475, 1.537, 1.57, 1.791, 2.33, 2.77},
00973    {4.115, 3.995, 4.345, 4.915, 5.667, 6.583, 7.547, 8.4, 8.806,
```

00974 8.416, 7.274, 5.963, 4.777, 3.806, 2.936, 2.114, 1.555, 1.194,  
00975 0.9788, 0.8251, 0.7444, 0.7323, 0.8067, 0.4073, 0.4371, 0.8223, 1.294,  
00976 1.799, 1.845, 1.715, 1.83, 2.164, 2.718, 3.019},  
00977 {4.267, 4.151, 4.545, 5.138, 5.928, 6.9, 7.906, 8.818, 9.276,  
00978 8.888, 7.7, 6.275, 4.923, 3.83, 2.949, 2.126, 1.57, 1.211,  
00979 0.9919, 0.8349, 0.7611, 0.7613, 0.8757, 0.4158, 0.6205, 0.9285, 1.474,  
00980 1.966, 1.886, 1.546, 1.534, 1.58, 1.727, 1.781},  
00981 {4.357, 4.249, 4.661, 5.37, 6.245, 7.245, 8.255, 9.142, 9.591,  
00982 9.187, 7.974, 6.489, 5.106, 3.983, 3.055, 2.223, 1.646, 1.263,  
00983 1.021, 0.8595, 0.7835, 0.7906, 0.9308, 0.4752, 0.6857, 1.015, 1.644,  
00984 2.238, 2.142, 1.682, 1.628, 1.619, 1.797, 1.869},  
00985 {4.506, 4.431, 4.893, 5.628, 6.535, 7.541, 8.516, 9.354, 9.751,  
00986 9.305, 8.052, 6.578, 5.209, 4.113, 3.227, 2.379, 1.745, 1.32,  
00987 1.053, 0.8816, 0.796, 0.7899, 0.9061, 1.191, 0.5672, 0.9669, 1.349,  
00988 1.746, 1.888, 1.592, 1.62, 1.791, 2.222, 2.485},  
00989 {4.576, 4.506, 4.983, 5.745, 6.664, 7.67, 8.657, 9.446, 9.792,  
00990 9.306, 8.032, 6.569, 5.251, 4.219, 3.374, 2.515, 1.835, 1.359,  
00991 1.057, 0.8733, 0.7654, 0.7338, 0.8346, 1.067, 0.6038, 1.054, 1.38,  
00992 1.684, 1.756, 1.589, 1.679, 2.018, 2.626, 2.986},  
00993 {4.675, 4.625, 5.078, 5.828, 6.714, 7.661, 8.588, 9.4, 9.761,  
00994 9.26, 7.973, 6.455, 5.133, 4.127, 3.339, 2.562, 1.884, 1.37,  
00995 1.036, 0.8316, 0.7014, 0.6314, 0.6329, 0.6673, 0.7434, 1.216, 1.142,  
00996 1.346, 1.515, 1.614, 1.773, 2.186, 2.758, 3.13},  
00997 {4.664, 4.649, 5.144, 5.923, 6.826, 7.78, 8.69, 9.369, 9.701,  
00998 9.297, 8.02, 6.479, 5.147, 4.209, 3.474, 2.764, 2.093, 1.519,  
00999 1.113, 0.8486, 0.683, 0.5849, 0.5324, 0.5209, 0.5327, 1.058, 0.9241,  
01000 1.048, 1.287, 1.427, 1.546, 1.782, 2.111, 2.301},  
01001 {4.771, 4.822, 5.408, 6.232, 7.155, 8.095, 8.92, 9.55, 9.74,  
01002 9.14, 7.776, 6.287, 5.065, 4.198, 3.518, 2.824, 2.145, 1.537,  
01003 1.096, 0.8121, 0.644, 0.5446, 0.4895, 0.4683, 0.463, 0.4501, 0.773,  
01004 0.7994, 0.9657, 1.138, 1.222, 1.365, 1.564, 1.666},  
01005 {5.238, 5.493, 6.287, 7.287, 8.326, 9.282, 10.02, 10.41, 10.22,  
01006 9.194, 7.64, 6.172, 5.058, 4.272, 3.57, 2.766, 1.962, 1.313,  
01007 0.8882, 0.6342, 0.4953, 0.4159, 0.3845, 0.3668, 0.3524, 0.3383, 0.6226,  
01008 0.6524, 0.702, 0.662, 0.6453, 0.6304, 0.6111, 0.5462},  
01009 {5.459, 5.786, 6.717, 8.018, 9.426, 10.72, 11.67, 11.98, 11.43,  
01010 9.923, 8.082, 6.51, 5.425, 4.719, 3.958, 2.937, 1.953, 1.26,  
01011 0.8432, 0.5868, 0.4389, 0.3576, 0.3281, 0.3182, 0.296, 0.2263, 0.5208,  
01012 0.5264, 0.5272, 0.439, 0.4076, 0.3774, 0.3392, 0.2954},  
01013 {{1.93, 2.082, 2.236, 2.401, 2.486, 2.46, 2.242, 1.936, 1.632,  
01014 1.309, 1.205, 0.996, 0.8843, 0.5832, 0.3788, 0.2472, 0.1935, 0.199,  
01015 0.2177, 0.3668, 0.2468, 0.1727, 0.1235, 0.1211, 0.09577, 0.05738, 0.01593,  
01016 0.01572, 0.01585, 0.01477, 0.01213, 0.01077, 0.008684, 0.005934},  
01017 {2.406, 2.637, 3.006, 3.467, 3.941, 4.339, 4.464, 4.221, 3.692,  
01018 2.961, 2.333, 1.856, 1.579, 1.321, 0.9877, 0.7954, 0.535, 0.3953,  
01019 0.3269, 0.3153, 0.4016, 0.4948, 0.4946, 0.3969, 0.2986, 0.157, 0.08327,  
01020 0.09294, 0.1047, 0.09143, 0.08129, 0.06982, 0.05108, 0.0324},  
01021 {2.891, 3.082, 3.531, 4.114, 4.759, 5.387, 5.81, 5.856, 5.447,  
01022 4.602, 3.716, 2.967, 2.422, 1.997, 1.543, 1.312, 1.043, 0.7202,  
01023 0.5009, 0.3828, 0.3369, 0.3204, 0.3053, 0.2956, 0.2344, 0.3256, 0.2033,  
01024 0.2183, 0.2574, 0.252, 0.2454, 0.2496, 0.2494, 0.2289},  
01025 {3.257, 3.412, 3.896, 4.558, 5.307, 6.077, 6.732, 7.047, 6.849,  
01026 6.037, 4.935, 3.973, 3.242, 2.755, 2.32, 1.807, 1.313, 0.9215,  
01027 0.6619, 0.4999, 0.3902, 0.3134, 0.2686, 0.245, 0.2368, 0.2048, 0.1813,  
01028 0.2732, 0.3298, 0.3568, 0.3526, 0.3493, 0.3549, 0.3469},  
01029 {3.587, 3.682, 4.173, 4.839, 5.622, 6.472, 7.24, 7.722, 7.706,  
01030 6.99, 5.826, 4.706, 3.861, 3.255, 2.675, 1.997, 1.417, 0.9866,  
01031 0.7187, 0.561, 0.4546, 0.3832, 0.3527, 0.3395, 0.3279, 0.4213, 0.3649,  
01032 0.4532, 0.5745, 0.6214, 0.6234, 0.6274, 0.6214, 0.5786},  
01033 {3.86, 3.88, 4.313, 4.965, 5.741, 6.605, 7.439, 8.066, 8.231,  
01034 7.669, 6.514, 5.29, 4.308, 3.595, 2.953, 2.22, 1.613, 1.142,  
01035 0.8371, 0.6574, 0.5366, 0.4574, 0.4295, 0.4016, 0.3794, 0.5616, 0.5829,  
01036 0.7168, 0.9521, 1.025, 1.053, 1.129, 1.25, 1.317},  
01037 {4.081, 4.041, 4.427, 5.032, 5.788, 6.668, 7.53, 8.225, 8.513,  
01038 8.083, 6.997, 5.729, 4.642, 3.826, 3.135, 2.369, 1.741, 1.266,  
01039 0.9467, 0.7487, 0.6194, 0.5505, 0.6097, 0.7323, 0.5351, 0.6646, 0.7098,  
01040 0.9842, 1.318, 1.49, 1.534, 1.718, 2.035, 2.285},  
01041 {4.216, 4.126, 4.503, 5.084, 5.827, 6.712, 7.64, 8.42, 8.793,  
01042 8.425, 7.34, 6.028, 4.838, 3.895, 3.074, 2.264, 1.681, 1.272,  
01043 1.009, 0.8308, 0.7345, 0.7299, 0.8734, 1.142, 0.5401, 0.7387, 0.9717,  
01044 1.328, 1.652, 1.667, 1.69, 1.954, 2.622, 3.19},  
01045 {4.272, 4.151, 4.513, 5.089, 5.851, 6.773, 7.683, 8.504, 8.944,  
01046 8.636, 7.572, 6.215, 4.931, 3.883, 2.974, 2.147, 1.591, 1.231,  
01047 1.009, 0.859, 0.7919, 0.8024, 0.896, 0.4226, 0.5516, 0.8571, 1.354,  
01048 1.883, 1.918, 1.778, 1.904, 2.264, 2.865, 3.19},  
01049 {4.268, 4.149, 4.512, 5.115, 5.895, 6.826, 7.773, 8.616, 9.068,  
01050 8.77, 7.691, 6.307, 4.976, 3.859, 2.932, 2.123, 1.578, 1.222,  
01051 0.9995, 0.8488, 0.7895, 0.8146, 0.9335, 0.4125, 0.6071, 0.932, 1.493,  
01052 1.986, 1.896, 1.55, 1.538, 1.583, 1.731, 1.792},  
01053 {4.24, 4.152, 4.565, 5.21, 6.012, 6.947, 7.876, 8.687, 9.116,  
01054 8.769, 7.647, 6.266, 4.953, 3.871, 2.972, 2.151, 1.594, 1.22,  
01055 0.9831, 0.8291, 0.7694, 0.7899, 0.9141, 0.4431, 0.6319, 0.9495, 1.544,  
01056 2.116, 2.045, 1.603, 1.547, 1.538, 1.698, 1.778},  
01057 {4.22, 4.152, 4.587, 5.265, 6.061, 6.961, 7.879, 8.672, 9.045,  
01058 8.614, 7.444, 6.083, 4.845, 3.844, 2.996, 2.197, 1.608, 1.21,  
01059 0.9591, 0.8021, 0.7294, 0.7305, 0.8376, 1.102, 0.495, 0.808, 1.134,  
01060 1.49, 1.689, 1.458, 1.475, 1.619, 1.978, 2.214},

```

01061 {4.139, 4.115, 4.593, 5.262, 6.061, 6.95, 7.815, 8.557, 8.84,
01062 8.311, 7.1, 5.777, 4.638, 3.742, 2.972, 2.201, 1.594, 1.167,
01063 0.9052, 0.7436, 0.6504, 0.62, 0.7047, 0.8275, 0.483, 0.7578, 0.9831,
01064 1.246, 1.432, 1.377, 1.433, 1.677, 2.116, 2.395},
01065 {4.027, 4.03, 4.521, 5.195, 5.986, 6.839, 7.642, 8.241, 8.387,
01066 7.78, 6.581, 5.331, 4.311, 3.538, 2.878, 2.173, 1.57, 1.118,
01067 0.8368, 0.6691, 0.5655, 0.5116, 0.5213, 0.5546, 0.5962, 0.7812, 0.7265,
01068 0.8823, 1.107, 1.292, 1.385, 1.633, 2.016, 2.214},
01069 {3.787, 3.852, 4.369, 5.07, 5.845, 6.625, 7.28, 7.735, 7.755,
01070 7.082, 5.9, 4.755, 3.857, 3.202, 2.64, 2.031, 1.478, 1.036,
01071 0.7546, 0.5863, 0.4836, 0.422, 0.3872, 0.3827, 0.4058, 0.6138, 0.5067,
01072 0.6048, 0.7857, 0.9735, 1.052, 1.172, 1.322, 1.358},
01073 {3.556, 3.717, 4.24, 4.935, 5.651, 6.309, 6.815, 7.1, 6.959,
01074 6.199, 5.099, 4.12, 3.39, 2.875, 2.37, 1.798, 1.291, 0.8979,
01075 0.6433, 0.4909, 0.3991, 0.3437, 0.3063, 0.2936, 0.2983, 0.283, 0.3393,
01076 0.3825, 0.4825, 0.6185, 0.6697, 0.7099, 0.7581, 0.7492},
01077 {3.344, 3.611, 4.169, 4.812, 5.45, 5.983, 6.257, 6.263, 5.844,
01078 4.989, 4.049, 3.337, 2.825, 2.403, 1.893, 1.346, 0.9248, 0.6385,
01079 0.4618, 0.3493, 0.281, 0.2397, 0.2114, 0.1989, 0.1936, 0.1739, 0.1941,
01080 0.2137, 0.2382, 0.2613, 0.2565, 0.2526, 0.2392, 0.2172},
01081 {3.617, 4.108, 4.785, 5.409, 5.873, 6.058, 5.747, 5.269, 4.577,
01082 3.776, 3.142, 2.721, 2.304, 1.764, 1.195, 0.7421, 0.4648, 0.3052,
01083 0.2132, 0.1552, 0.1201, 0.1028, 0.09496, 0.09272, 0.092, 0.06578, 0.09663,
01084 0.1039, 0.1161, 0.1127, 0.1081, 0.1019, 0.09228, 0.08144}},
01085 {{4.776, 5.263, 6.105, 7.209, 8.284, 9.138, 9.553, 9.22, 8.161,
01086 6.644, 5.129, 4.002, 3.207, 2.842, 2.466, 1.792, 1.171, 0.7381,
01087 0.5022, 0.4253, 0.4401, 0.5454, 0.7785, 0.6418, 0.4899, 0.3887, 0.3859,
01088 0.2463, 0.248, 0.1831, 0.1414, 0.1233, 0.104, 0.07015},
01089 {4.315, 4.602, 5.253, 6.073, 6.933, 7.732, 8.306, 8.428, 7.938,
01090 6.819, 5.489, 4.358, 3.446, 2.895, 2.476, 1.885, 1.473, 1.07,
01091 0.7373, 0.5219, 0.4618, 0.454, 0.4569, 0.4263, 0.397, 0.4683, 0.4728,
01092 0.4348, 0.4638, 0.3711, 0.3221, 0.2786, 0.2347, 0.153},
01093 {4.034, 4.197, 4.772, 5.576, 6.46, 7.358, 8.131, 8.568, 8.457,
01094 7.592, 6.31, 5.095, 4.065, 3.363, 2.711, 2.026, 1.686, 1.33,
01095 1.006, 0.7144, 0.5379, 0.4588, 0.4427, 0.451, 0.4738, 0.5938, 0.595,
01096 0.5555, 0.6354, 0.6081, 0.5877, 0.5953, 0.6217, 0.6394},
01097 {4.052, 4.154, 4.699, 5.472, 6.377, 7.353, 8.242, 8.879, 9.016,
01098 8.382, 7.119, 5.777, 4.673, 3.861, 3.197, 2.408, 1.717, 1.19,
01099 0.8643, 0.6598, 0.5193, 0.4314, 0.3937, 0.3863, 0.4011, 0.3591, 0.4105,
01100 0.4673, 0.5539, 0.5717, 0.5531, 0.5518, 0.5563, 0.5379},
01101 {4.153, 4.213, 4.735, 5.466, 6.334, 7.3, 8.25, 8.998, 9.286,
01102 8.779, 7.558, 6.165, 5, 4.154, 3.419, 2.575, 1.849, 1.298,
01103 0.9519, 0.7439, 0.6103, 0.537, 0.545, 0.5816, 0.5683, 0.7049, 0.6064,
01104 0.7504, 0.9181, 0.9021, 0.8857, 0.8707, 0.8466, 0.7728},
01105 {4.248, 4.254, 4.738, 5.442, 6.269, 7.19, 8.14, 8.939, 9.34,
01106 8.978, 7.82, 6.411, 5.184, 4.273, 3.525, 2.664, 1.942, 1.394,
01107 1.033, 0.8143, 0.6732, 0.5954, 0.6154, 0.6352, 0.7198, 0.8339, 0.8119,
01108 1.046, 1.311, 1.244, 1.264, 1.359, 1.508, 1.629},
01109 {4.354, 4.315, 4.761, 5.42, 6.219, 7.126, 8.036, 8.844, 9.254,
01110 8.926, 7.833, 6.444, 5.174, 4.22, 3.457, 2.639, 1.948, 1.432,
01111 1.087, 0.8685, 0.7349, 0.6908, 0.8, 0.9447, 0.6798, 0.8977, 1.108,
01112 1.425, 1.666, 1.47, 1.449, 1.545, 1.768, 1.943},
01113 {4.362, 4.274, 4.672, 5.312, 6.096, 7.012, 7.964, 8.785, 9.19,
01114 8.837, 7.774, 6.356, 5.026, 3.988, 3.137, 2.353, 1.747, 1.332,
01115 1.065, 0.8964, 0.8178, 0.823, 0.9551, 1.158, 0.5854, 0.9884, 1.483,
01116 2.035, 2.091, 1.603, 1.521, 1.55, 1.723, 1.854},
01117 {4.3, 4.174, 4.537, 5.167, 5.965, 6.89, 7.821, 8.663, 9.083,
01118 8.735, 7.634, 6.25, 4.936, 3.878, 2.975, 2.165, 1.611, 1.244,
01119 1.01, 0.864, 0.8216, 0.8541, 0.947, 0.4242, 0.4811, 0.9771, 1.569,
01120 2.131, 2.042, 1.616, 1.566, 1.597, 1.77, 1.859},
01121 {4.191, 4.082, 4.474, 5.086, 5.853, 6.754, 7.678, 8.491, 8.887,
01122 8.503, 7.408, 6.052, 4.775, 3.739, 2.846, 2.066, 1.531, 1.182,
01123 0.9574, 0.8217, 0.7847, 0.8129, 0.9004, 0.3845, 0.5912, 0.9488, 1.54,
01124 2.073, 1.987, 1.591, 1.544, 1.55, 1.68, 1.746},
01125 {4.067, 4, 4.384, 4.981, 5.736, 6.618, 7.486, 8.26, 8.602,
01126 8.169, 7.063, 5.748, 4.553, 3.579, 2.743, 1.997, 1.466, 1.117,
01127 0.8947, 0.756, 0.7085, 0.7246, 0.8226, 0.3804, 0.4137, 0.8378, 1.338,
01128 1.831, 1.823, 1.452, 1.396, 1.406, 1.546, 1.607},
01129 {3.852, 3.824, 4.228, 4.825, 5.559, 6.404, 7.233, 7.922, 8.179,
01130 7.678, 6.56, 5.348, 4.269, 3.383, 2.62, 1.89, 1.369, 1.029,
01131 0.8128, 0.6781, 0.613, 0.6018, 0.6705, 0.6956, 0.4126, 0.6338, 0.8946,
01132 1.186, 1.405, 1.278, 1.277, 1.398, 1.661, 1.83},
01133 {3.577, 3.6, 4.019, 4.632, 5.336, 6.114, 6.899, 7.485, 7.599,
01134 6.995, 5.898, 4.813, 3.907, 3.169, 2.502, 1.82, 1.298, 0.9403,
01135 0.7213, 0.5834, 0.4987, 0.4571, 0.4767, 0.4929, 0.357, 0.5342, 0.6218,
01136 0.8321, 1.072, 1.133, 1.182, 1.326, 1.548, 1.687},
01137 {3.251, 3.338, 3.79, 4.396, 5.094, 5.833, 6.465, 6.856, 6.763,
01138 6.045, 5.018, 4.111, 3.394, 2.814, 2.265, 1.671, 1.175, 0.8195,
01139 0.6074, 0.4777, 0.3953, 0.3477, 0.3295, 0.3281, 0.3202, 0.3996, 0.4085,
01140 0.5264, 0.6945, 0.8446, 0.9062, 1.002, 1.145, 1.182},
01141 {2.844, 3.014, 3.474, 4.053, 4.689, 5.317, 5.776, 5.907, 5.606,
01142 4.864, 3.98, 3.263, 2.726, 2.292, 1.824, 1.308, 0.8985, 0.62,
01143 0.4532, 0.3508, 0.2879, 0.2496, 0.2268, 0.217, 0.2089, 0.1723, 0.2384,
01144 0.3033, 0.3816, 0.457, 0.4738, 0.4945, 0.522, 0.511},
01145 {2.32, 2.558, 2.981, 3.486, 3.981, 4.383, 4.505, 4.35, 3.92,
01146 3.298, 2.688, 2.238, 1.901, 1.574, 1.197, 0.8142, 0.5528, 0.3893,
01147 0.2874, 0.2207, 0.179, 0.153, 0.1353, 0.126, 0.118, 0.0996, 0.1038,

```

01148 0.1254, 0.1556, 0.1673, 0.1637, 0.1606, 0.1599, 0.1512},  
01149 {1.604, 1.812, 2.085, 2.33, 2.517, 2.565, 2.341, 2.07, 1.768,  
01150 1.479, 1.254, 1.072, 0.8657, 0.6304, 0.4269, 0.2773, 0.1887, 0.1365,  
01151 0.104, 0.08106, 0.06679, 0.05742, 0.0504, 0.04538, 0.04064, 0.03211,  
01152 0.02611,  
01153 0.02727, 0.03066, 0.03125, 0.02913, 0.02667, 0.02485, 0.02217},  
01154 {0.4429, 0.4652, 0.4579, 0.4357, 0.388, 0.3401, 0.2901, 0.2551, 0.2192,  
01155 0.1788, 0.1348, 0.0914, 0.05502, 0.03148, 0.01858, 0.01216, 0.009078,  
01156 0.007534,  
01157 0.006492, 0.006257, 0.006301, 0.006115, 0.005378, 0.005084, 0.005022,  
01158 0.002766, 0.00246,  
01159 0.001431, 0.001208, 0.0014, 0.001257, 0.001091, 0.0009076, 0.0005632}},  
01160 {{6.159, 6.267, 7.137, 8.441, 9.868, 11.32, 12.68, 13.44, 13.31,  
01161 11.97, 9.899, 7.982, 6.529, 5.456, 4.582, 3.411, 2.447, 1.735,  
01162 1.201, 0.8902, 0.7385, 0.82, 0.8247, 0.6792, 0.5978, 0.7594, 1.311,  
01163 0.8001, 0.7222, 0.5196, 0.4, 0.3513, 0.3049, 0.2033},  
01164 {6.025, 6.218, 7.108, 8.305, 9.599, 10.86, 11.89, 12.5, 12.4,  
01165 11.24, 9.372, 7.562, 6.173, 5.198, 4.359, 3.187, 2.293, 1.674,  
01166 1.209, 0.8758, 0.6848, 0.7086, 0.7544, 0.6892, 0.6462, 0.8934, 1.25,  
01167 0.9268, 0.9078, 0.7087, 0.6172, 0.5399, 0.4671, 0.304},  
01168 {5.423, 5.508, 6.112, 6.997, 8.017, 9.107, 10.07, 10.75, 10.95,  
01169 10.26, 8.775, 7.139, 5.746, 4.723, 3.902, 2.898, 2.14, 1.579,  
01170 1.155, 0.86, 0.6747, 0.5805, 0.5689, 0.5958, 0.6918, 0.8863, 1.028,  
01171 0.8369, 0.8899, 0.8223, 0.7867, 0.7493, 0.7297, 0.6911},  
01172 {5.03, 5.007, 5.533, 6.337, 7.269, 8.306, 9.353, 10.16, 10.53,  
01173 10.09, 8.802, 7.176, 5.772, 4.785, 3.999, 3.116, 2.292, 1.613,  
01174 1.161, 0.8825, 0.7044, 0.602, 0.5675, 0.5799, 0.633, 0.5335, 0.75,  
01175 0.782, 0.8887, 0.8566, 0.8053, 0.779, 0.7512, 0.6894},  
01176 {4.854, 4.795, 5.239, 5.991, 6.89, 7.916, 8.949, 9.794, 10.25,  
01177 9.932, 8.787, 7.214, 5.806, 4.824, 4.046, 3.151, 2.318, 1.656,  
01178 1.216, 0.9337, 0.7505, 0.6429, 0.6093, 0.6241, 0.6313, 0.828, 0.8414,  
01179 1.007, 1.191, 1.083, 1.037, 1.002, 0.963, 0.8665},  
01180 {4.689, 4.616, 5.08, 5.801, 6.661, 7.626, 8.619, 9.471, 9.972,  
01181 9.711, 8.582, 7.039, 5.65, 4.664, 3.923, 3.04, 2.236, 1.63,  
01182 1.223, 0.9546, 0.7831, 0.6819, 0.6534, 0.6599, 0.5823, 0.9021, 1.016,  
01183 1.267, 1.495, 1.395, 1.398, 1.508, 1.711, 1.859},  
01184 {4.575, 4.466, 4.925, 5.634, 6.479, 7.412, 8.343, 9.218, 9.74,  
01185 9.468, 8.319, 6.801, 5.4, 4.37, 3.571, 2.713, 2.014, 1.498,  
01186 1.158, 0.9364, 0.7954, 0.7273, 0.7681, 0.8566, 0.6687, 0.9463, 1.219,  
01187 1.575, 1.788, 1.546, 1.525, 1.635, 1.887, 2.09},  
01188 {4.408, 4.299, 4.735, 5.433, 6.275, 7.22, 8.175, 9.032, 9.47,  
01189 9.097, 7.942, 6.45, 5.08, 4.019, 3.16, 2.338, 1.735, 1.324,  
01190 1.063, 0.9095, 0.8421, 0.8249, 0.8269, 0.6041, 0.6079, 1.07, 1.601,  
01191 2.169, 2.204, 1.68, 1.593, 1.624, 1.805, 1.939},  
01192 {4.26, 4.136, 4.516, 5.172, 5.992, 6.946, 7.925, 8.739, 9.102,  
01193 8.667, 7.534, 6.13, 4.865, 3.837, 2.924, 2.126, 1.572, 1.207,  
01194 0.9693, 0.8391, 0.8016, 0.8213, 0.8584, 0.4114, 0.632, 0.9986, 1.59,  
01195 2.132, 2.045, 1.617, 1.567, 1.595, 1.754, 1.846},  
01196 {4.068, 3.944, 4.321, 4.953, 5.759, 6.696, 7.598, 8.349, 8.636,  
01197 8.134, 6.966, 5.681, 4.531, 3.592, 2.726, 1.981, 1.454, 1.112,  
01198 0.8863, 0.7642, 0.7336, 0.7503, 0.7883, 0.3649, 0.5742, 0.9245, 1.482,  
01199 1.979, 1.91, 1.537, 1.492, 1.497, 1.609, 1.655},  
01200 {3.784, 3.702, 4.08, 4.686, 5.436, 6.274, 7.107, 7.772, 7.978,  
01201 7.457, 6.38, 5.207, 4.171, 3.317, 2.572, 1.856, 1.347, 1.012,  
01202 0.8031, 0.6877, 0.6459, 0.6441, 0.6934, 0.3392, 0.5146, 0.7711, 1.206,  
01203 1.627, 1.668, 1.358, 1.31, 1.315, 1.411, 1.432},  
01204 {3.39, 3.409, 3.832, 4.448, 5.159, 5.932, 6.66, 7.149, 7.206,  
01205 6.618, 5.602, 4.608, 3.743, 3.01, 2.347, 1.689, 1.206, 0.8923,  
01206 0.6966, 0.5763, 0.5136, 0.4878, 0.5216, 0.5783, 0.3499, 0.515, 0.7012,  
01207 0.9131, 1.167, 1.133, 1.139, 1.212, 1.359, 1.445},  
01208 {3.031, 3.122, 3.551, 4.115, 4.781, 5.496, 6.101, 6.433, 6.32,  
01209 5.654, 4.707, 3.886, 3.211, 2.629, 2.053, 1.473, 1.024, 0.7318,  
01210 0.5579, 0.445, 0.3748, 0.3356, 0.3272, 0.3261, 0.3502, 0.4067, 0.4482,  
01211 0.5625, 0.7534, 0.8328, 0.8615, 0.9261, 1.038, 1.075},  
01212 {2.556, 2.697, 3.11, 3.64, 4.251, 4.887, 5.363, 5.492, 5.176,  
01213 4.453, 3.662, 3.064, 2.599, 2.164, 1.677, 1.161, 0.7816, 0.5445,  
01214 0.4076, 0.3171, 0.258, 0.2227, 0.2043, 0.1946, 0.1903, 0.2423, 0.2411,  
01215 0.2984, 0.3661, 0.4305, 0.4483, 0.4735, 0.5096, 0.5082},  
01216 {1.982, 2.163, 2.522, 2.962, 3.444, 3.894, 4.12, 3.996, 3.538,  
01217 2.915, 2.39, 2.044, 1.761, 1.418, 1.026, 0.6684, 0.4452, 0.3147,  
01218 0.2354, 0.1814, 0.1474, 0.1272, 0.1136, 0.1042, 0.09334, 0.07244, 0.09453,  
01219 0.1067, 0.1323, 0.1309, 0.1255, 0.1235, 0.1251, 0.1207},  
01220 {1.313, 1.48, 1.706, 1.932, 2.113, 2.193, 2.081, 1.804, 1.487,  
01221 1.196, 0.9808, 0.8365, 0.6791, 0.4931, 0.3304, 0.2112, 0.1439, 0.1054,  
01222 0.08052, 0.06314, 0.05248, 0.04667, 0.0419, 0.03731, 0.03192, 0.02135,  
01223 0.01682,  
01224 0.0156, 0.01767, 0.01723, 0.0161, 0.01526, 0.0148, 0.01411},  
01225 {0.242, 0.2311, 0.2162, 0.1962, 0.1752, 0.1604, 0.1387, 0.1112, 0.08183,  
01226 0.05815, 0.04045, 0.02676, 0.01677, 0.01075, 0.007653, 0.005984, 0.00512,  
01227 0.004795,  
01228 0.004786, 0.004999, 0.004952, 0.004352, 0.003443, 0.002664, 0.002223,  
01229 0.001163, 0.001542,  
01230 0.0002821, 0.0001951, 0.000206, 0.0001656, 0.0001206, 8.303e-05,  
01231 5.901e-05},  
01232 {0.0001232, 0.0001559, 0.0001539, 0.0001693, 0.0002134, 0.0002031,  
01233 0.0001037, 1.126e-05, 5.382e-06,  
01234 1.867e-06, 5.983e-07, 2.464e-07, 1.576e-07, 1.322e-07, 1.312e-07,

```

01235     1.319e-07, 3.921e-07, 3.583e-06,
01236     3.815e-05, 6.754e-05, 0.0001004, 0.0002135, 0.0004217, 0.0007681,
01237     0.001524, 0.0004274, 0.000876,
01238     2.698e-05, 1.328e-12, 1.445e-13, 9.798e-14, 8.583e-14, 9.786e-14,
01239     1.774e-13}},
01240     {{6.595, 6.532, 7.313, 8.453, 9.864, 11.47, 13.06, 14.28, 14.67,
01241     13.68, 11.56, 9.275, 7.452, 6.201, 5.275, 4.16, 2.898, 2.003,
01242     1.4, 1.04, 0.7754, 0.7071, 0.7598, 0.799, 0.825, 0.9217, 1.851,
01243     1.254, 1.138, 0.8159, 0.6311, 0.5427, 0.4614, 0.3814},
01244     {6.516, 6.556, 7.327, 8.526, 9.924, 11.42, 12.85, 13.82, 14.03,
01245     13.05, 11.03, 8.863, 7.108, 5.878, 4.956, 3.797, 2.704, 1.92,
01246     1.344, 0.9685, 0.7276, 0.6364, 0.6746, 0.7239, 0.786, 0.9333, 1.793,
01247     1.344, 1.234, 0.8885, 0.7949, 0.6932, 0.5878, 0.4871},
01248     {6.179, 6.202, 6.853, 7.807, 8.924, 10.13, 11.21, 12.01, 12.29,
01249     11.63, 10.05, 8.152, 6.536, 5.386, 4.503, 3.473, 2.521, 1.809,
01250     1.273, 0.9058, 0.6837, 0.5774, 0.5746, 0.6269, 0.7726, 0.9434, 1.275,
01251     1.102, 1.148, 0.9922, 0.9195, 0.8713, 0.8162, 0.7358},
01252     {5.401, 5.302, 5.812, 6.634, 7.64, 8.785, 9.902, 10.82, 11.3,
01253     10.96, 9.696, 7.981, 6.412, 5.281, 4.41, 3.469, 2.606, 1.892,
01254     1.37, 1.034, 0.8087, 0.6766, 0.6565, 0.6981, 0.7901, 0.6904, 1.01,
01255     1.062, 1.192, 1.063, 1.016, 0.9639, 0.8911, 0.7914},
01256     {5.101, 4.973, 5.426, 6.18, 7.138, 8.24, 9.32, 10.29, 10.9,
01257     10.75, 9.665, 8.035, 6.469, 5.319, 4.452, 3.502, 2.649, 1.941,
01258     1.431, 1.09, 0.869, 0.7456, 0.7339, 0.7833, 0.8079, 1.059, 1.104,
01259     1.303, 1.515, 1.253, 1.185, 1.131, 1.076, 0.9437},
01260     {4.936, 4.795, 5.272, 5.985, 6.878, 7.91, 8.989, 9.922, 10.53,
01261     10.37, 9.278, 7.698, 6.176, 5.044, 4.178, 3.263, 2.472, 1.849,
01262     1.402, 1.087, 0.8859, 0.7846, 0.8226, 0.8854, 0.9635, 1.037, 1.251,
01263     1.527, 1.706, 1.5, 1.503, 1.644, 1.914, 2.113},
01264     {4.796, 4.617, 5.024, 5.703, 6.591, 7.617, 8.632, 9.544, 10.07,
01265     9.749, 8.552, 6.983, 5.55, 4.462, 3.573, 2.707, 2.021, 1.537,
01266     1.216, 1.017, 0.9039, 0.8702, 0.9836, 1.21, 0.6125, 1.009, 1.311,
01267     1.688, 1.862, 1.575, 1.568, 1.696, 2.001, 2.214},
01268     {4.522, 4.356, 4.742, 5.465, 6.36, 7.357, 8.359, 9.269, 9.706,
01269     9.237, 7.95, 6.476, 5.137, 4.086, 3.214, 2.373, 1.75, 1.33,
01270     1.071, 0.9379, 0.8929, 0.9071, 0.9736, 1.305, 0.5218, 1.054, 1.605,
01271     2.105, 1.976, 1.563, 1.521, 1.56, 1.765, 1.875},
01272     {4.201, 4.084, 4.453, 5.134, 5.998, 7.007, 8.042, 8.894, 9.218,
01273     8.665, 7.393, 5.966, 4.728, 3.77, 2.956, 2.16, 1.585, 1.199,
01274     0.9637, 0.8579, 0.8414, 0.8686, 0.8189, 1.154, 0.4693, 0.9934, 1.568,
01275     2.075, 1.962, 1.563, 1.524, 1.545, 1.704, 1.786},
01276     {3.87, 3.761, 4.135, 4.74, 5.547, 6.523, 7.533, 8.287, 8.542,
01277     7.978, 6.743, 5.463, 4.36, 3.491, 2.739, 1.993, 1.453, 1.095,
01278     0.8767, 0.7822, 0.7664, 0.777, 0.8145, 1.109, 0.4094, 0.8854, 1.413,
01279     1.91, 1.872, 1.47, 1.421, 1.428, 1.538, 1.583},
01280     {3.565, 3.517, 3.908, 4.525, 5.299, 6.159, 6.982, 7.581, 7.734,
01281     7.15, 6.028, 4.918, 3.993, 3.242, 2.541, 1.833, 1.321, 0.9862,
01282     0.7851, 0.6877, 0.6504, 0.6409, 0.6657, 0.7916, 0.3852, 0.627, 0.8774,
01283     1.306, 1.713, 1.397, 1.317, 1.308, 1.379, 1.377},
01284     {3.27, 3.307, 3.718, 4.324, 5.008, 5.72, 6.391, 6.82, 6.844,
01285     6.25, 5.256, 4.321, 3.562, 2.929, 2.309, 1.67, 1.183, 0.8581,
01286     0.6613, 0.5437, 0.4817, 0.4549, 0.4828, 0.4971, 0.343, 0.4517, 0.5928,
01287     0.7482, 1.114, 1.156, 1.127, 1.142, 1.266, 1.325},
01288     {2.881, 2.972, 3.365, 3.885, 4.479, 5.095, 5.612, 5.869, 5.739,
01289     5.109, 4.233, 3.497, 2.928, 2.45, 1.923, 1.37, 0.937, 0.6588,
01290     0.4974, 0.3913, 0.3216, 0.2799, 0.263, 0.2476, 0.2702, 0.3664, 0.3897,
01291     0.4754, 0.6181, 0.6968, 0.7144, 0.7507, 0.8199, 0.8256},
01292     {2.352, 2.522, 2.914, 3.377, 3.888, 4.391, 4.73, 4.773, 4.456,
01293     3.814, 3.103, 2.576, 2.19, 1.824, 1.372, 0.9129, 0.606, 0.4281,
01294     0.3241, 0.25, 0.1992, 0.1685, 0.1489, 0.1316, 0.116, 0.1598, 0.1448,
01295     0.1805, 0.2224, 0.2379, 0.2369, 0.2454, 0.2679, 0.2718},
01296     {1.666, 1.833, 2.135, 2.486, 2.847, 3.14, 3.202, 3.006, 2.612,
01297     2.127, 1.726, 1.486, 1.277, 0.9733, 0.6654, 0.4233, 0.2852, 0.2051,
01298     0.1537, 0.1174, 0.09422, 0.08017, 0.06975, 0.06009, 0.04775, 0.03319,
01299     0.03371,
01300     0.03896, 0.04544, 0.04203, 0.03927, 0.03814, 0.03917, 0.04012},
01301     {0.8975, 0.9719, 1.03, 1.066, 1.034, 0.9374, 0.7957, 0.662, 0.5656,
01302     0.4856, 0.4141, 0.3239, 0.2283, 0.1478, 0.09439, 0.06056, 0.04188,
01303     0.03179,
01304     0.02625, 0.02293, 0.02134, 0.01999, 0.01796, 0.01508, 0.01136, 0.006243,
01305     0.005399,
01306     0.002554, 0.002671, 0.002877, 0.002693, 0.002456, 0.002169, 0.001592},
01307     {0.005568, 0.003081, 0.001936, 0.001388, 0.001138, 0.0009141, 0.0003913,
01308     7.042e-05, 1.305e-05,
01309     9.014e-06, 5.819e-06, 3.047e-06, 1.303e-06, 5.602e-07, 2.183e-07,
01310     1.757e-07, 3.825e-07, 2.566e-06,
01311     1.334e-05, 1.436e-05, 1.976e-05, 7.261e-05, 0.0002657, 0.0005962,
01312     0.001653, 0.0002773, 0.0008521,
01313     1.309e-06, 3.72e-14, 2.315e-16, 2.404e-15, 7.283e-17, 5.816e-17,
01314     1.165e-16},
01315     {5.606e-05, 7.174e-05, 7.065e-05, 8.779e-05, 0.0001175, 0.0001418,
01316     6.181e-05, 7.462e-06, 8.135e-06,
01317     6.922e-06, 3.21e-06, 1.063e-06, 3.185e-07, 7.307e-08, 1.298e-08,
01318     1.751e-08, 6.792e-08, 5.277e-07,
01319     7.612e-06, 1.832e-05, 4.78e-05, 0.0001019, 0.0001703, 0.0003801, 0.001213,
01320     0.0002105, 0.0006011,
01321     2.875e-06, 7.798e-13, 1.214e-13, 8.329e-14, 7.553e-14, 1.014e-13,

```



```

01322     1.901e-13}}
01323 };
01324
01325 #ifdef _OPENACC
01326 #pragma acc declare copyin(clim_oh_var)
01327 #endif
01328
01329 double clim_oh(
01330     double t,
01331     double lat,
01332     double p) {
01333
01334     /* Get seconds since begin of year... */
01335     double sec = FMOD(t, 365.25 * 86400.);
01336     while (sec < 0)
01337         sec += 365.25 * 86400.;
01338
01339     /* Check pressure... */
01340     if (p < clim_oh_ps[0])
01341         p = clim_oh_ps[0];
01342     else if (p > clim_oh_ps[33])
01343         p = clim_oh_ps[33];
01344
01345     /* Check latitude... */
01346     if (lat < clim_oh_lats[0])
01347         lat = clim_oh_lats[0];
01348     else if (lat > clim_oh_lats[17])
01349         lat = clim_oh_lats[17];
01350
01351     /* Get indices... */
01352     int isec = locate_irr(clim_oh_secs, 12, sec);
01353     int ilat = locate_reg(clim_oh_lats, 18, lat);
01354     int ip = locate_irr(clim_oh_ps, 34, p);
01355
01356     /* Interpolate OH climatology (Pommrich et al., 2014)... */
01357     double aux00 = LIN(clim_oh_ps[ip],
01358         clim_oh_var[isec][ilat][ip],
01359         clim_oh_ps[ip + 1],
01360         clim_oh_var[isec][ilat][ip + 1], p);
01361     double aux01 = LIN(clim_oh_ps[ip],
01362         clim_oh_var[isec][ilat + 1][ip],
01363         clim_oh_ps[ip + 1],
01364         clim_oh_var[isec][ilat + 1][ip + 1], p);
01365     double aux10 = LIN(clim_oh_ps[ip],
01366         clim_oh_var[isec + 1][ilat][ip],
01367         clim_oh_ps[ip + 1],
01368         clim_oh_var[isec + 1][ilat][ip + 1], p);
01369     double aux11 = LIN(clim_oh_ps[ip],
01370         clim_oh_var[isec + 1][ilat + 1][ip],
01371         clim_oh_ps[ip + 1],
01372         clim_oh_var[isec + 1][ilat + 1][ip + 1], p);
01373     aux00 = LIN(clim_oh_lats[ilat], aux00, clim_oh_lats[ilat + 1], aux01, lat);
01374     aux11 = LIN(clim_oh_lats[ilat], aux10, clim_oh_lats[ilat + 1], aux11, lat);
01375     aux00 = LIN(clim_oh_secs[isec], aux00, clim_oh_secs[isec + 1], aux11, sec);
01376     return GSL_MAX(1e6 * aux00, 0.0);
01377 }
01378
01379 /*****
01380
01381 static double clim_tropo_secs[12] = {
01382     1209600.00, 3888000.00, 6393600.00,
01383     9072000.00, 11664000.00, 14342400.00,
01384     16934400.00, 19612800.00, 22291200.00,
01385     24883200.00, 27561600.00, 30153600.00
01386 };
01387
01388 #ifdef _OPENACC
01389 #pragma acc declare copyin(clim_tropo_secs)
01390 #endif
01391
01392 static double clim_tropo_lats[73]
01393 = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
01394     -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
01395     -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
01396     -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
01397     15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
01398     45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
01399     75, 77.5, 80, 82.5, 85, 87.5, 90
01400 };
01401
01402 #ifdef _OPENACC
01403 #pragma acc declare copyin(clim_tropo_lats)
01404 #endif
01405
01406 static double clim_tropo_tps[12][73]
01407 = { { 324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
01408     297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,

```

```

01409      175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
01410      99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
01411      98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
01412      152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
01413      277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
01414      275.3, 275.6, 275.4, 274.1, 273.5},
01415 {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
01416      300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
01417      150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
01418      98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
01419      98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
01420      220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
01421      284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01422      287.5, 286.2, 285.8},
01423 {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
01424      297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
01425      161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
01426      100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01427      99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
01428      186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
01429      279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
01430      304.3, 304.9, 306, 306.6, 306.2, 306},
01431 {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
01432      290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
01433      195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
01434      102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
01435      99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
01436      148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
01437      263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
01438      315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
01439 {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
01440      260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
01441      205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
01442      101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
01443      102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
01444      165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
01445      273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01446      325.3, 325.8, 325.8},
01447 {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
01448      222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
01449      228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
01450      105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
01451      106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
01452      127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
01453      251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
01454      308.5, 312.2, 313.1, 313.3},
01455 {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
01456      187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
01457      235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
01458      110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
01459      111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
01460      117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
01461      224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
01462      275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
01463 {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01464      185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
01465      233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
01466      110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
01467      112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
01468      120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
01469      230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
01470      278.2, 282.6, 287.4, 290.9, 292.5, 293},
01471 {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
01472      183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
01473      243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
01474      114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
01475      110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
01476      114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
01477      203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
01478      276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
01479 {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
01480      215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
01481      237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
01482      111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01483      106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
01484      112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
01485      206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01486      279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
01487      305.1},
01488 {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01489      253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
01490      223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
01491      108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
01492      102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
01493      109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
01494      241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
01495      286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},

```

```

01496 {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
01497 284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
01498 175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
01499 100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
01500 100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
01501 186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
01502 280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
01503 281.7, 281.1, 281.2}
01504 };
01505
01506 #ifdef _OPENACC
01507 #pragma acc declare copyin(clim_tropo_tps)
01508 #endif
01509
01510 double clim_tropo(
01511     double t,
01512     double lat) {
01513
01514     /* Get seconds since begin of year... */
01515     double sec = FMOD(t, 365.25 * 86400.);
01516     while (sec < 0)
01517         sec += 365.25 * 86400.;
01518
01519     /* Get indices... */
01520     int isec = locate_irr(clim_tropo_secs, 12, sec);
01521     int ilat = locate_reg(clim_tropo_lats, 73, lat);
01522
01523     /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
01524     double p0 = LIN(clim_tropo_lats[ilat],
01525                     clim_tropo_tps[isec][ilat],
01526                     clim_tropo_lats[ilat + 1],
01527                     clim_tropo_tps[isec][ilat + 1], lat);
01528     double p1 = LIN(clim_tropo_lats[ilat],
01529                     clim_tropo_tps[isec + 1][ilat],
01530                     clim_tropo_lats[ilat + 1],
01531                     clim_tropo_tps[isec + 1][ilat + 1], lat);
01532     return LIN(clim_tropo_secs[isec], p0, clim_tropo_secs[isec + 1], p1, sec);
01533 }
01534
01535 /*****
01536
01537 void day2doy(
01538     int year,
01539     int mon,
01540     int day,
01541     int *doy) {
01542
01543     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01544     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01545
01546     /* Get day of year... */
01547     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
01548         *doy = d0l[mon - 1] + day - 1;
01549     else
01550         *doy = d0[mon - 1] + day - 1;
01551 }
01552
01553 /*****
01554
01555 void doy2day(
01556     int year,
01557     int doy,
01558     int *mon,
01559     int *day) {
01560
01561     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01562     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01563     int i;
01564
01565     /* Get month and day... */
01566     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
01567         for (i = 11; i >= 0; i--)
01568             if (d0l[i] <= doy)
01569                 break;
01570         *mon = i + 1;
01571         *day = doy - d0l[i] + 1;
01572     } else {
01573         for (i = 11; i >= 0; i--)
01574             if (d0[i] <= doy)
01575                 break;
01576         *mon = i + 1;
01577         *day = doy - d0[i] + 1;
01578     }
01579 }
01580
01581 /*****
01582

```

```

01583 void geo2cart(
01584     double z,
01585     double lon,
01586     double lat,
01587     double *x) {
01588
01589     double radius = z + RE;
01590     x[0] = radius * cos(lat / 180. * M_PI) * cos(lon / 180. * M_PI);
01591     x[1] = radius * cos(lat / 180. * M_PI) * sin(lon / 180. * M_PI);
01592     x[2] = radius * sin(lat / 180. * M_PI);
01593 }
01594
01595 /*****
01596
01597 void get_met(
01598     ctl_t * ctl,
01599     double t,
01600     met_t ** met0,
01601     met_t ** met1) {
01602
01603     static int init, ip, ix, iy;
01604
01605     met_t *mets;
01606
01607     char filename[LEN];
01608
01609     /* Set timer... */
01610     SELECT_TIMER("GET_MET", NVTX_READ);
01611
01612     /* Init... */
01613     if (t == ctl->t_start || !init) {
01614         init = 1;
01615
01616         get_met_help(t, -1, ctl->metbase, ctl->dt_met, filename);
01617         if (!read_met(ctl, filename, *met0))
01618             ERRMSG("Cannot open file!");
01619
01620         get_met_help(t + 1.0 * ctl->direction, 1, ctl->metbase, ctl->dt_met,
01621                     filename);
01622         if (!read_met(ctl, filename, *met1))
01623             ERRMSG("Cannot open file!");
01624 #ifdef _OPENACC
01625         met_t *met0up = *met0;
01626         met_t *met1up = *met1;
01627 #pragma acc update device(met0up[:1], met1up[:1])
01628 #endif
01629     }
01630
01631     /* Read new data for forward trajectories... */
01632     if (t > (*met1)->time && ctl->direction == 1) {
01633         mets = *met1;
01634         *met1 = *met0;
01635         *met0 = mets;
01636         get_met_help(t, 1, ctl->metbase, ctl->dt_met, filename);
01637         if (!read_met(ctl, filename, *met1))
01638             ERRMSG("Cannot open file!");
01639 #ifdef _OPENACC
01640         met_t *met1up = *met1;
01641 #pragma acc update device(met1up[:1])
01642 #endif
01643     }
01644
01645     /* Read new data for backward trajectories... */
01646     if (t < (*met0)->time && ctl->direction == -1) {
01647         mets = *met1;
01648         *met1 = *met0;
01649         *met0 = mets;
01650         get_met_help(t, -1, ctl->metbase, ctl->dt_met, filename);
01651         if (!read_met(ctl, filename, *met0))
01652             ERRMSG("Cannot open file!");
01653 #ifdef _OPENACC
01654         met_t *met0up = *met0;
01655 #pragma acc update device(met0up[:1])
01656 #endif
01657     }
01658
01659     /* Check that grids are consistent... */
01660     if ((*met0)->nx != (*met1)->nx
01661         || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
01662         ERRMSG("Meteo grid dimensions do not match!");
01663     for (ix = 0; ix < (*met0)->nx; ix++)
01664         if (fabs((*met0)->lon[ix] - (*met1)->lon[ix]) > 0.001)
01665             ERRMSG("Meteo grid longitudes do not match!");
01666     for (iy = 0; iy < (*met0)->ny; iy++)
01667         if (fabs((*met0)->lat[iy] - (*met1)->lat[iy]) > 0.001)
01668             ERRMSG("Meteo grid latitudes do not match!");
01669     for (ip = 0; ip < (*met0)->np; ip++)

```

```

01670     if (fabs((*met0)->p[ip] - (*met1)->p[ip]) > 0.001)
01671         ERRMSG("Meteo grid pressure levels do not match!");
01672 }
01673
01674 /*****
01675
01676 void get_met_help(
01677     double t,
01678     int direct,
01679     char *metbase,
01680     double dt_met,
01681     char *filename) {
01682     char repl[LEN];
01683
01684     double t6, r;
01685
01686     int year, mon, day, hour, min, sec;
01687
01688     /* Round time to fixed intervals... */
01689     if (direct == -1)
01690         t6 = floor(t / dt_met) * dt_met;
01691     else
01692         t6 = ceil(t / dt_met) * dt_met;
01693
01694     /* Decode time... */
01695     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
01696
01697     /* Set filename... */
01698     sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
01699     sprintf(repl, "%d", year);
01700     get_met_replace(filename, "YYYY", repl);
01701     sprintf(repl, "%02d", mon);
01702     get_met_replace(filename, "MM", repl);
01703     sprintf(repl, "%02d", day);
01704     get_met_replace(filename, "DD", repl);
01705     sprintf(repl, "%02d", hour);
01706     get_met_replace(filename, "HH", repl);
01707 }
01708
01709 /*****
01710
01711 void get_met_replace(
01712     char *orig,
01713     char *search,
01714     char *repl) {
01715     char buffer[LEN], *ch;
01716
01717     /* Iterate... */
01718     for (int i = 0; i < 3; i++) {
01719         /* Replace sub-string... */
01720         if (!(ch = strstr(orig, search)))
01721             return;
01722         strncpy(buffer, orig, (size_t) (ch - orig));
01723         buffer[ch - orig] = 0;
01724         sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
01725         orig[0] = 0;
01726         strcpy(orig, buffer);
01727     }
01728 }
01729
01730 /*****
01731
01732 void intpol_met_space_3d(
01733     met_t *met,
01734     float array[EX][EY][EP],
01735     double p,
01736     double lon,
01737     double lat,
01738     double *var,
01739     int *ci,
01740     double *cw,
01741     int init) {
01742     /* Check longitude... */
01743     if (met->lon[met->nx - 1] > 180 && lon < 0)
01744         lon += 360;
01745
01746     /* Get interpolation indices and weights... */
01747     if (init) {
01748         ci[0] = locate_irr(met->p, met->np, p);
01749         ci[1] = locate_reg(met->lon, met->nx, lon);
01750         ci[2] = locate_reg(met->lat, met->ny, lat);
01751         cw[0] = (met->p[ci[0] + 1] - p)
01752             / (met->p[ci[0] + 1] - met->p[ci[0]]);

```

```

01757     cw[1] = (met->lon[ci[1] + 1] - lon)
01758     / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01759     cw[2] = (met->lat[ci[2] + 1] - lat)
01760     / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01761 }
01762
01763 /* Interpolate vertically... */
01764 double aux00 =
01765     cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
01766     + array[ci[1]][ci[2]][ci[0] + 1];
01767 double aux01 =
01768     cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
01769         array[ci[1]][ci[2] + 1][ci[0] + 1])
01770     + array[ci[1]][ci[2] + 1][ci[0] + 1];
01771 double aux10 =
01772     cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
01773         array[ci[1] + 1][ci[2]][ci[0] + 1])
01774     + array[ci[1] + 1][ci[2]][ci[0] + 1];
01775 double aux11 =
01776     cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
01777         array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
01778     + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];
01779
01780 /* Interpolate horizontally... */
01781 aux00 = cw[2] * (aux00 - aux01) + aux01;
01782 aux11 = cw[2] * (aux10 - aux11) + aux11;
01783 *var = cw[1] * (aux00 - aux11) + aux11;
01784 }
01785
01786
01787 /*****
01788
01789 void intpol_met_space_2d(
01790     met_t * met,
01791     float array[EX][EY],
01792     double lon,
01793     double lat,
01794     double *var,
01795     int *ci,
01796     double *cw,
01797     int init) {
01798
01799     /* Check longitude... */
01800     if (met->lon[met->nx - 1] > 180 && lon < 0)
01801         lon += 360;
01802
01803     /* Get interpolation indices and weights... */
01804     if (init) {
01805         ci[1] = locate_reg(met->lon, met->nx, lon);
01806         ci[2] = locate_reg(met->lat, met->ny, lat);
01807         cw[1] = (met->lon[ci[1] + 1] - lon)
01808             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01809         cw[2] = (met->lat[ci[2] + 1] - lat)
01810             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01811     }
01812
01813     /* Set variables... */
01814     double aux00 = array[ci[1]][ci[2]];
01815     double aux01 = array[ci[1]][ci[2] + 1];
01816     double aux10 = array[ci[1] + 1][ci[2]];
01817     double aux11 = array[ci[1] + 1][ci[2] + 1];
01818
01819     /* Interpolate horizontally... */
01820     if (isfinite(aux00) && isfinite(aux01))
01821         aux00 = cw[2] * (aux00 - aux01) + aux01;
01822     else if (cw[2] < 0.5)
01823         aux00 = aux01;
01824     if (isfinite(aux10) && isfinite(aux11))
01825         aux11 = cw[2] * (aux10 - aux11) + aux11;
01826     else if (cw[2] > 0.5)
01827         aux11 = aux10;
01828     if (isfinite(aux00) && isfinite(aux11))
01829         *var = cw[1] * (aux00 - aux11) + aux11;
01830     else {
01831         if (cw[1] > 0.5)
01832             *var = aux00;
01833         else
01834             *var = aux11;
01835     }
01836 }
01837
01838 /*****
01839
01840 void intpol_met_time_3d(
01841     met_t * met0,
01842     float array0[EX][EY][EP],
01843     met_t * met1,

```

```

01844 float array1[EX][EY][EP],
01845 double ts,
01846 double p,
01847 double lon,
01848 double lat,
01849 double *var,
01850 int *ci,
01851 double *cw,
01852 int init) {
01853
01854 double var0, var1, wt;
01855
01856 /* Spatial interpolation... */
01857 intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
01858 intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, init);
01859
01860 /* Get weighting factor... */
01861 wt = (met1->time - ts) / (met1->time - met0->time);
01862
01863 /* Interpolate... */
01864 *var = wt * (var0 - var1) + var1;
01865 }
01866
01867 /*****
01868
01869 void intpol_met_time_2d(
01870     met_t * met0,
01871     float array0[EX][EY],
01872     met_t * met1,
01873     float array1[EX][EY],
01874     double ts,
01875     double lon,
01876     double lat,
01877     double *var,
01878     int *ci,
01879     double *cw,
01880     int init) {
01881
01882     double var0, var1, wt;
01883
01884     /* Spatial interpolation... */
01885     intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
01886     intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, init);
01887
01888     /* Get weighting factor... */
01889     wt = (met1->time - ts) / (met1->time - met0->time);
01890
01891     /* Interpolate... */
01892     *var = wt * (var0 - var1) + var1;
01893 }
01894
01895 /*****
01896
01897 void jsec2time(
01898     double jsec,
01899     int *year,
01900     int *mon,
01901     int *day,
01902     int *hour,
01903     int *min,
01904     int *sec,
01905     double *remain) {
01906
01907     struct tm t0, *t1;
01908
01909     t0.tm_year = 100;
01910     t0.tm_mon = 0;
01911     t0.tm_mday = 1;
01912     t0.tm_hour = 0;
01913     t0.tm_min = 0;
01914     t0.tm_sec = 0;
01915
01916     time_t jsec0 = (time_t) jsec + timegm(&t0);
01917     t1 = gmtime(&jsec0);
01918
01919     *year = t1->tm_year + 1900;
01920     *mon = t1->tm_mon + 1;
01921     *day = t1->tm_mday;
01922     *hour = t1->tm_hour;
01923     *min = t1->tm_min;
01924     *sec = t1->tm_sec;
01925     *remain = jsec - floor(jsec);
01926 }
01927
01928 /*****
01929
01930 double lapse_rate(

```

```

01931     double t,
01932     double h2o) {
01933
01934     /*
01935      * Calculate moist adiabatic lapse rate [K/km] from temperature [K]
01936      * and water vapor volume mixing ratio [1].
01937      *
01938      * Reference: https://en.wikipedia.org/wiki/Lapse\_rate
01939      */
01940     const double a = RA * SQR(t), r = SH(h2o) / (1. - SH(h2o));
01941
01942     return 1e3 * G0 * (a + LV * r * t) / (CPD * a + SQR(LV) * r * EPS);
01943 }
01944
01945 /*****
01946
01947
01948 int locate_irr(
01949     double *xx,
01950     int n,
01951     double x) {
01952
01953     int ilo = 0;
01954     int ihi = n - 1;
01955     int i = (ihi + ilo) >> 1;
01956
01957     if (xx[i] < xx[i + 1])
01958         while (ihi > ilo + 1) {
01959             i = (ihi + ilo) >> 1;
01960             if (xx[i] > x)
01961                 ihi = i;
01962             else
01963                 ilo = i;
01964         } else
01965         while (ihi > ilo + 1) {
01966             i = (ihi + ilo) >> 1;
01967             if (xx[i] <= x)
01968                 ihi = i;
01969             else
01970                 ilo = i;
01971         }
01972
01973     return ilo;
01974 }
01975
01976 /*****
01977
01978 int locate_reg(
01979     double *xx,
01980     int n,
01981     double x) {
01982
01983     /* Calculate index... */
01984     int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
01985
01986     /* Check range... */
01987     if (i < 0)
01988         i = 0;
01989     else if (i >= n - 2)
01990         i = n - 2;
01991
01992     return i;
01993 }
01994
01995 /*****
01996
01997 double nat_temperature(
01998     double p,
01999     double h2o,
02000     double hno3) {
02001
02002     double p_hno3 = hno3 * p / 1.333224;
02003     double p_h2o = h2o * p / 1.333224;
02004     double a = 0.009179 - 0.00088 * log10(p_h2o);
02005     double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
02006     double c = -11397.0 / a;
02007     double tnat = (-b + sqrt(b * b - 4. * c)) / 2.;
02008     double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
02009     if (x2 > 0)
02010         tnat = x2;
02011
02012     return tnat;
02013 }
02014
02015 /*****
02016
02017 int read_atm(

```



```

02018     const char *filename,
02019     ctl_t * ctl,
02020     atm_t * atm) {
02021
02022     FILE *in;
02023
02024     char line[LEN], *tok;
02025
02026     double t0;
02027
02028     int dimid, ip, iq, ncid, varid;
02029
02030     size_t nparts;
02031
02032     /* Set timer... */
02033     SELECT_TIMER("READ_ATM", NVTX_READ);
02034
02035     /* Init... */
02036     atm->np = 0;
02037
02038     /* Write info... */
02039     printf("Read atmospheric data: %s\n", filename);
02040
02041     /* Read ASCII data... */
02042     if (ctl->atm_type == 0) {
02043
02044         /* Open file... */
02045         if (!(in = fopen(filename, "r"))) {
02046             WARN("File not found!");
02047             return 0;
02048         }
02049
02050         /* Read line... */
02051         while (fgets(line, LEN, in)) {
02052
02053             /* Read data... */
02054             TOK(line, tok, "%lg", atm->time[atm->np]);
02055             TOK(NULL, tok, "%lg", atm->p[atm->np]);
02056             TOK(NULL, tok, "%lg", atm->lon[atm->np]);
02057             TOK(NULL, tok, "%lg", atm->lat[atm->np]);
02058             for (iq = 0; iq < ctl->nq; iq++)
02059                 TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
02060
02061             /* Convert altitude to pressure... */
02062             atm->p[atm->np] = P(atm->p[atm->np]);
02063
02064             /* Increment data point counter... */
02065             if (++atm->np > NP)
02066                 ERRMSG("Too many data points!");
02067         }
02068
02069         /* Close file... */
02070         fclose(in);
02071     }
02072
02073     /* Read binary data... */
02074     else if (ctl->atm_type == 1) {
02075
02076         /* Open file... */
02077         if (!(in = fopen(filename, "r")))
02078             return 0;
02079
02080         /* Read data... */
02081         FREAD(&atm->np, int,
02082             1,
02083             in);
02084         FREAD(atm->time, double,
02085             (size_t) atm->np,
02086             in);
02087         FREAD(atm->p, double,
02088             (size_t) atm->np,
02089             in);
02090         FREAD(atm->lon, double,
02091             (size_t) atm->np,
02092             in);
02093         FREAD(atm->lat, double,
02094             (size_t) atm->np,
02095             in);
02096         for (iq = 0; iq < ctl->nq; iq++)
02097             FREAD(atm->q[iq], double,
02098                 (size_t) atm->np,
02099                 in);
02100
02101         /* Close file... */
02102         fclose(in);
02103     }
02104

```

```

02105  /* Read netCDF data... */
02106  else if (ctl->atm_type == 2) {
02107
02108      /* Open file... */
02109      if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02110          return 0;
02111
02112      /* Get dimensions... */
02113      NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
02114      NC(nc_inq_dimlen(ncid, dimid, &nparts));
02115      atm->np = (int) nparts;
02116      if (atm->np > NP)
02117          ERRMSG("Too many particles!");
02118
02119      /* Get time... */
02120      NC(nc_inq_varid(ncid, "time", &varid));
02121      NC(nc_get_var_double(ncid, varid, &t0));
02122      for (ip = 0; ip < atm->np; ip++)
02123          atm->time[ip] = t0;
02124
02125      /* Read geolocations... */
02126      NC(nc_inq_varid(ncid, "PRESS", &varid));
02127      NC(nc_get_var_double(ncid, varid, atm->p));
02128      NC(nc_inq_varid(ncid, "LON", &varid));
02129      NC(nc_get_var_double(ncid, varid, atm->lon));
02130      NC(nc_inq_varid(ncid, "LAT", &varid));
02131      NC(nc_get_var_double(ncid, varid, atm->lat));
02132
02133      /* Read variables... */
02134      if (ctl->qnt_p >= 0)
02135          if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
02136              NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
02137      if (ctl->qnt_t >= 0)
02138          if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
02139              NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
02140      if (ctl->qnt_u >= 0)
02141          if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
02142              NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
02143      if (ctl->qnt_v >= 0)
02144          if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
02145              NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
02146      if (ctl->qnt_w >= 0)
02147          if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
02148              NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
02149      if (ctl->qnt_h2o >= 0)
02150          if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
02151              NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
02152      if (ctl->qnt_o3 >= 0)
02153          if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
02154              NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
02155      if (ctl->qnt_theta >= 0)
02156          if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
02157              NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
02158      if (ctl->qnt_pv >= 0)
02159          if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
02160              NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
02161
02162      /* Check data... */
02163      for (ip = 0; ip < atm->np; ip++)
02164          if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
02165              || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
02166              || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
02167              || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
02168              || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
02169          atm->time[ip] = GSL_NAN;
02170          atm->p[ip] = GSL_NAN;
02171          atm->lon[ip] = GSL_NAN;
02172          atm->lat[ip] = GSL_NAN;
02173          for (iq = 0; iq < atm->nq; iq++)
02174              atm->q[iq][ip] = GSL_NAN;
02175          } else {
02176              if (ctl->qnt_h2o >= 0)
02177                  atm->q[ctl->qnt_h2o][ip] *= 1.608;
02178              if (ctl->qnt_pv >= 0)
02179                  atm->q[ctl->qnt_pv][ip] *= 1e6;
02180              if (atm->lon[ip] > 180)
02181                  atm->lon[ip] -= 360;
02182          }
02183
02184      /* Close file... */
02185      NC(nc_close(ncid));
02186  }
02187
02188  /* Error... */
02189  else
02190      ERRMSG("Atmospheric data type not supported!");
02191

```

```

02192  /* Check number of points... */
02193  if (atm->np < 1)
02194      ERRMSG("Can not read any data!");
02195
02196  /* Return success... */
02197  return 1;
02198 }
02199
02200 /*****
02201
02202 void read_ctl(
02203     const char *filename,
02204     int argc,
02205     char *argv[],
02206     ctl_t * ctl) {
02207
02208     /* Set timer... */
02209     SELECT_TIMER("READ_CTL", NVTX_READ);
02210
02211     /* Write info... */
02212     printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
02213           "(executable: %s | compiled: %s, %s)\n\n",
02214           argv[0], __DATE__, __TIME__);
02215
02216     /* Initialize quantity indices... */
02217     ctl->qnt_ens = -1;
02218     ctl->qnt_m = -1;
02219     ctl->qnt_r = -1;
02220     ctl->qnt_rho = -1;
02221     ctl->qnt_ps = -1;
02222     ctl->qnt_ts = -1;
02223     ctl->qnt_zs = -1;
02224     ctl->qnt_us = -1;
02225     ctl->qnt_vs = -1;
02226     ctl->qnt_pt = -1;
02227     ctl->qnt_tt = -1;
02228     ctl->qnt_zt = -1;
02229     ctl->qnt_h2ot = -1;
02230     ctl->qnt_z = -1;
02231     ctl->qnt_p = -1;
02232     ctl->qnt_t = -1;
02233     ctl->qnt_u = -1;
02234     ctl->qnt_v = -1;
02235     ctl->qnt_w = -1;
02236     ctl->qnt_h2o = -1;
02237     ctl->qnt_o3 = -1;
02238     ctl->qnt_lwc = -1;
02239     ctl->qnt_iwc = -1;
02240     ctl->qnt_pc = -1;
02241     ctl->qnt_cl = -1;
02242     ctl->qnt_plcl = -1;
02243     ctl->qnt_plfc = -1;
02244     ctl->qnt_pel = -1;
02245     ctl->qnt_cape = -1;
02246     ctl->qnt_hno3 = -1;
02247     ctl->qnt_oh = -1;
02248     ctl->qnt_psat = -1;
02249     ctl->qnt_psice = -1;
02250     ctl->qnt_pw = -1;
02251     ctl->qnt_sh = -1;
02252     ctl->qnt_rh = -1;
02253     ctl->qnt_rhice = -1;
02254     ctl->qnt_theta = -1;
02255     ctl->qnt_tvirt = -1;
02256     ctl->qnt_lapse = -1;
02257     ctl->qnt_vh = -1;
02258     ctl->qnt_vz = -1;
02259     ctl->qnt_pv = -1;
02260     ctl->qnt_tdew = -1;
02261     ctl->qnt_tice = -1;
02262     ctl->qnt_tsts = -1;
02263     ctl->qnt_tnat = -1;
02264     ctl->qnt_stat = -1;
02265
02266     /* Read quantities... */
02267     ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
02268     if (ctl->nq > NQ)
02269         ERRMSG("Too many quantities!");
02270     for (int iq = 0; iq < ctl->nq; iq++) {
02271
02272         /* Read quantity name and format... */
02273         scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
02274         scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
02275                 ctl->qnt_format[iq]);
02276
02277         /* Try to identify quantity... */
02278         if (strcasecmp(ctl->qnt_name[iq], "ens") == 0) {

```

```

02279     ctl->qnt_ens = iq;
02280     sprintf(ctl->qnt_unit[iq], "-");
02281 } else if (strcasecmp(ctl->qnt_name[iq], "m") == 0) {
02282     ctl->qnt_m = iq;
02283     sprintf(ctl->qnt_unit[iq], "kg");
02284 } else if (strcasecmp(ctl->qnt_name[iq], "r") == 0) {
02285     ctl->qnt_r = iq;
02286     sprintf(ctl->qnt_unit[iq], "m");
02287 } else if (strcasecmp(ctl->qnt_name[iq], "rho") == 0) {
02288     ctl->qnt_rho = iq;
02289     sprintf(ctl->qnt_unit[iq], "kg/m^3");
02290 } else if (strcasecmp(ctl->qnt_name[iq], "ps") == 0) {
02291     ctl->qnt_ps = iq;
02292     sprintf(ctl->qnt_unit[iq], "hPa");
02293 } else if (strcasecmp(ctl->qnt_name[iq], "pt") == 0) {
02294     ctl->qnt_pt = iq;
02295     sprintf(ctl->qnt_unit[iq], "hPa");
02296 } else if (strcasecmp(ctl->qnt_name[iq], "tt") == 0) {
02297     ctl->qnt_tt = iq;
02298     sprintf(ctl->qnt_unit[iq], "K");
02299 } else if (strcasecmp(ctl->qnt_name[iq], "zt") == 0) {
02300     ctl->qnt_zt = iq;
02301     sprintf(ctl->qnt_unit[iq], "km");
02302 } else if (strcasecmp(ctl->qnt_name[iq], "h2ot") == 0) {
02303     ctl->qnt_h2ot = iq;
02304     sprintf(ctl->qnt_unit[iq], "ppv");
02305 } else if (strcasecmp(ctl->qnt_name[iq], "z") == 0) {
02306     ctl->qnt_z = iq;
02307     sprintf(ctl->qnt_unit[iq], "km");
02308 } else if (strcasecmp(ctl->qnt_name[iq], "p") == 0) {
02309     ctl->qnt_p = iq;
02310     sprintf(ctl->qnt_unit[iq], "hPa");
02311 } else if (strcasecmp(ctl->qnt_name[iq], "t") == 0) {
02312     ctl->qnt_t = iq;
02313     sprintf(ctl->qnt_unit[iq], "K");
02314 } else if (strcasecmp(ctl->qnt_name[iq], "u") == 0) {
02315     ctl->qnt_u = iq;
02316     sprintf(ctl->qnt_unit[iq], "m/s");
02317 } else if (strcasecmp(ctl->qnt_name[iq], "v") == 0) {
02318     ctl->qnt_v = iq;
02319     sprintf(ctl->qnt_unit[iq], "m/s");
02320 } else if (strcasecmp(ctl->qnt_name[iq], "w") == 0) {
02321     ctl->qnt_w = iq;
02322     sprintf(ctl->qnt_unit[iq], "hPa/s");
02323 } else if (strcasecmp(ctl->qnt_name[iq], "h2o") == 0) {
02324     ctl->qnt_h2o = iq;
02325     sprintf(ctl->qnt_unit[iq], "ppv");
02326 } else if (strcasecmp(ctl->qnt_name[iq], "o3") == 0) {
02327     ctl->qnt_o3 = iq;
02328     sprintf(ctl->qnt_unit[iq], "ppv");
02329 } else if (strcasecmp(ctl->qnt_name[iq], "lwc") == 0) {
02330     ctl->qnt_lwc = iq;
02331     sprintf(ctl->qnt_unit[iq], "kg/kg");
02332 } else if (strcasecmp(ctl->qnt_name[iq], "iwc") == 0) {
02333     ctl->qnt_iwc = iq;
02334     sprintf(ctl->qnt_unit[iq], "kg/kg");
02335 } else if (strcasecmp(ctl->qnt_name[iq], "pc") == 0) {
02336     ctl->qnt_pc = iq;
02337     sprintf(ctl->qnt_unit[iq], "hPa");
02338 } else if (strcasecmp(ctl->qnt_name[iq], "cl") == 0) {
02339     ctl->qnt_cl = iq;
02340     sprintf(ctl->qnt_unit[iq], "kg/m^2");
02341 } else if (strcasecmp(ctl->qnt_name[iq], "plcl") == 0) {
02342     ctl->qnt_plcl = iq;
02343     sprintf(ctl->qnt_unit[iq], "hPa");
02344 } else if (strcasecmp(ctl->qnt_name[iq], "plfc") == 0) {
02345     ctl->qnt_plfc = iq;
02346     sprintf(ctl->qnt_unit[iq], "hPa");
02347 } else if (strcasecmp(ctl->qnt_name[iq], "pel") == 0) {
02348     ctl->qnt_pel = iq;
02349     sprintf(ctl->qnt_unit[iq], "hPa");
02350 } else if (strcasecmp(ctl->qnt_name[iq], "cape") == 0) {
02351     ctl->qnt_cape = iq;
02352     sprintf(ctl->qnt_unit[iq], "J/kg");
02353 } else if (strcasecmp(ctl->qnt_name[iq], "hno3") == 0) {
02354     ctl->qnt_hno3 = iq;
02355     sprintf(ctl->qnt_unit[iq], "ppv");
02356 } else if (strcasecmp(ctl->qnt_name[iq], "oh") == 0) {
02357     ctl->qnt_oh = iq;
02358     sprintf(ctl->qnt_unit[iq], "molec/cm^3");
02359 } else if (strcasecmp(ctl->qnt_name[iq], "psat") == 0) {
02360     ctl->qnt_psat = iq;
02361     sprintf(ctl->qnt_unit[iq], "hPa");
02362 } else if (strcasecmp(ctl->qnt_name[iq], "psice") == 0) {
02363     ctl->qnt_psice = iq;
02364     sprintf(ctl->qnt_unit[iq], "hPa");
02365 } else if (strcasecmp(ctl->qnt_name[iq], "pw") == 0) {

```

```

02366     ctl->qnt_pw = iq;
02367     sprintf(ctl->qnt_unit[iq], "hPa");
02368 } else if (strcasecmp(ctl->qnt_name[iq], "sh") == 0) {
02369     ctl->qnt_sh = iq;
02370     sprintf(ctl->qnt_unit[iq], "kg/kg");
02371 } else if (strcasecmp(ctl->qnt_name[iq], "rh") == 0) {
02372     ctl->qnt_rh = iq;
02373     sprintf(ctl->qnt_unit[iq], "%%");
02374 } else if (strcasecmp(ctl->qnt_name[iq], "rhice") == 0) {
02375     ctl->qnt_rhice = iq;
02376     sprintf(ctl->qnt_unit[iq], "%%");
02377 } else if (strcasecmp(ctl->qnt_name[iq], "theta") == 0) {
02378     ctl->qnt_theta = iq;
02379     sprintf(ctl->qnt_unit[iq], "K");
02380 } else if (strcasecmp(ctl->qnt_name[iq], "tvirt") == 0) {
02381     ctl->qnt_tvirt = iq;
02382     sprintf(ctl->qnt_unit[iq], "K");
02383 } else if (strcasecmp(ctl->qnt_name[iq], "lapse") == 0) {
02384     ctl->qnt_lapse = iq;
02385     sprintf(ctl->qnt_unit[iq], "K/km");
02386 } else if (strcasecmp(ctl->qnt_name[iq], "vh") == 0) {
02387     ctl->qnt_vh = iq;
02388     sprintf(ctl->qnt_unit[iq], "m/s");
02389 } else if (strcasecmp(ctl->qnt_name[iq], "vz") == 0) {
02390     ctl->qnt_vz = iq;
02391     sprintf(ctl->qnt_unit[iq], "m/s");
02392 } else if (strcasecmp(ctl->qnt_name[iq], "pv") == 0) {
02393     ctl->qnt_pv = iq;
02394     sprintf(ctl->qnt_unit[iq], "PVU");
02395 } else if (strcasecmp(ctl->qnt_name[iq], "tdew") == 0) {
02396     ctl->qnt_tdew = iq;
02397     sprintf(ctl->qnt_unit[iq], "K");
02398 } else if (strcasecmp(ctl->qnt_name[iq], "tice") == 0) {
02399     ctl->qnt_tice = iq;
02400     sprintf(ctl->qnt_unit[iq], "K");
02401 } else if (strcasecmp(ctl->qnt_name[iq], "tsts") == 0) {
02402     ctl->qnt_tsts = iq;
02403     sprintf(ctl->qnt_unit[iq], "K");
02404 } else if (strcasecmp(ctl->qnt_name[iq], "tnat") == 0) {
02405     ctl->qnt_tnat = iq;
02406     sprintf(ctl->qnt_unit[iq], "K");
02407 } else if (strcasecmp(ctl->qnt_name[iq], "stat") == 0) {
02408     ctl->qnt_stat = iq;
02409     sprintf(ctl->qnt_unit[iq], "-");
02410 } else
02411     scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
02412 }
02413
02414 /* Time steps of simulation... */
02415 ctl->direction =
02416     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
02417 if (ctl->direction != -1 && ctl->direction != 1)
02418     ERRMSG("Set DIRECTION to -1 or 1!");
02419 ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
02420 ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "180", NULL);
02421
02422 /* Meteorological data... */
02423 scan_ctl(filename, argc, argv, "METBASE", -1, "-", ctl->metbase);
02424 ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
02425 ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
02426 ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
02427 ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
02428 if (ctl->met_dx < 1 || ctl->met_dy < 1 || ctl->met_dp < 1)
02429     ERRMSG("MET_DX, MET_DY, and MET_DP need to be greater than zero!");
02430 ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
02431 ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
02432 ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
02433 if (ctl->met_sx < 1 || ctl->met_sy < 1 || ctl->met_sp < 1)
02434     ERRMSG("MET_SX, MET_SY, and MET_SP need to be greater than zero!");
02435 ctl->met_detrend =
02436     scan_ctl(filename, argc, argv, "MET_DETREND", -1, "-999", NULL);
02437 ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
02438 if (ctl->met_np > EP)
02439     ERRMSG("Too many levels!");
02440 for (int ip = 0; ip < ctl->met_np; ip++)
02441     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
02442 ctl->met_geopot_sx
02443     = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SX", -1, "6", NULL);
02444 ctl->met_geopot_sy
02445     = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SY", -1, "4", NULL);
02446 if (ctl->met_geopot_sx < 1 || ctl->met_geopot_sy < 1)
02447     ERRMSG("MET_GEOPOT_SX and MET_GEOPOT_SY need to be greater than zero!");
02448 ctl->met_tropo =
02449     (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
02450 ctl->met_dt_out =
02451     scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
02452

```

```

02453  /* Isosurface parameters... */
02454  ctl->isosurf =
02455      (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
02456  scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
02457
02458  /* Diffusion parameters... */
02459  ctl->turb_dx_trop =
02460      scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
02461  ctl->turb_dx_strat =
02462      scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
02463  ctl->turb_dz_trop =
02464      scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
02465  ctl->turb_dz_strat =
02466      scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
02467  ctl->turb_mesox =
02468      scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
02469  ctl->turb_mesoz =
02470      scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
02471
02472  /* Convection... */
02473  ctl->conv_cape =
02474      scan_ctl(filename, argc, argv, "CONV_CAPE", -1, "-999", NULL);
02475
02476  /* Species parameters... */
02477  scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
02478  if (strcasecmp(ctl->species, "SO2") == 0) {
02479      ctl->molmass = 64.066;
02480      ctl->oh_chem[0] = 3.3e-31; /* (JPL Publication 15-10) */
02481      ctl->oh_chem[1] = 4.3; /* (JPL Publication 15-10) */
02482      ctl->oh_chem[2] = 1.6e-12; /* (JPL Publication 15-10) */
02483      ctl->oh_chem[3] = 0.0; /* (JPL Publication 15-10) */
02484      ctl->wet_depo[2] = 1.3e-2; /* (Sander, 2015) */
02485      ctl->wet_depo[3] = 2900.0; /* (Sander, 2015) */
02486      ctl->wet_depo[6] = 1.3e-2; /* (Sander, 2015) */
02487      ctl->wet_depo[7] = 2900.0; /* (Sander, 2015) */
02488  } else {
02489      ctl->molmass =
02490          scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
02491      ctl->tdec_trop =
02492          scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
02493      ctl->tdec_strat =
02494          scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
02495      for (int ip = 0; ip < 4; ip++)
02496          ctl->oh_chem[ip] =
02497              scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
02498      for (int ip = 0; ip < 1; ip++)
02499          ctl->dry_depo[ip] =
02500              scan_ctl(filename, argc, argv, "DRY_DEPO", ip, "0", NULL);
02501      for (int ip = 0; ip < 8; ip++)
02502          ctl->wet_depo[ip] =
02503              scan_ctl(filename, argc, argv, "WET_DEPO", ip, "0", NULL);
02504  }
02505
02506  /* PSC analysis... */
02507  ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
02508  ctl->psc_hno3 =
02509      scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
02510
02511  /* Output of atmospheric data... */
02512  scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->atm_basename);
02513  scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
02514  ctl->atm_dt_out =
02515      scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
02516  ctl->atm_filter =
02517      (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
02518  ctl->atm_stride =
02519      (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
02520  ctl->atm_type =
02521      (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
02522
02523  /* Output of CSI data... */
02524  scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->csi_basename);
02525  ctl->csi_dt_out =
02526      scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
02527  scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->csi_obsfile);
02528  ctl->csi_obsmin =
02529      scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
02530  ctl->csi_modmin =
02531      scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
02532  ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
02533  ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
02534  ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
02535  ctl->csi_lon0 =
02536      scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
02537  ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
02538  ctl->csi_nx =
02539      (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);

```

```

02540 ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
02541 ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
02542 ctl->csi_ny =
02543     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
02544
02545 /* Output of ensemble data... */
02546 scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->ens_basename);
02547
02548 /* Output of grid data... */
02549 scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
02550     ctl->grid_basename);
02551 scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->grid_gpfile);
02552 ctl->grid_dt_out =
02553     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
02554 ctl->grid_sparse =
02555     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
02556 ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
02557 ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
02558 ctl->grid_nz =
02559     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
02560 ctl->grid_lon0 =
02561     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
02562 ctl->grid_lon1 =
02563     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
02564 ctl->grid_nx =
02565     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
02566 ctl->grid_lat0 =
02567     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
02568 ctl->grid_lat1 =
02569     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
02570 ctl->grid_ny =
02571     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
02572
02573 /* Output of profile data... */
02574 scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
02575     ctl->prof_basename);
02576 scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->prof_obsfile);
02577 ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
02578 ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
02579 ctl->prof_nz =
02580     (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
02581 ctl->prof_lon0 =
02582     scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
02583 ctl->prof_lon1 =
02584     scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
02585 ctl->prof_nx =
02586     (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
02587 ctl->prof_lat0 =
02588     scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
02589 ctl->prof_lat1 =
02590     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
02591 ctl->prof_ny =
02592     (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
02593
02594 /* Output of sample data... */
02595 scan_ctl(filename, argc, argv, "SAMPLE_BASENAME", -1, "-",
02596     ctl->sample_basename);
02597 scan_ctl(filename, argc, argv, "SAMPLE_OBSFILE", -1, "-",
02598     ctl->sample_obsfile);
02599 ctl->sample_dx =
02600     scan_ctl(filename, argc, argv, "SAMPLE_DX", -1, "50", NULL);
02601 ctl->sample_dz =
02602     scan_ctl(filename, argc, argv, "SAMPLE_DZ", -1, "-999", NULL);
02603
02604 /* Output of station data... */
02605 scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
02606     ctl->stat_basename);
02607 ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
02608 ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
02609 ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
02610 }
02611
02612 /*****
02613
02614 int read_met(
02615     ctl_t * ctl,
02616     char *filename,
02617     met_t * met) {
02618
02619     int ncid;
02620
02621     /* Write info... */
02622     printf("Read meteorological data: %s\n", filename);
02623
02624     /* Open netCDF file... */
02625     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
02626         WARN("File not found!");

```

```

02627     return 0;
02628 }
02629
02630 /* Read coordinates of meteorological data... */
02631 read_met_grid(filename, ncid, ctl, met);
02632
02633 /* Read meteo data on vertical levels... */
02634 read_met_levels(ncid, ctl, met);
02635
02636 /* Extrapolate data for lower boundary... */
02637 read_met_extrapolate(met);
02638
02639 /* Read surface data... */
02640 read_met_surface(ncid, met);
02641
02642 /* Create periodic boundary conditions... */
02643 read_met_periodic(met);
02644
02645 /* Downsampling... */
02646 read_met_sample(ctl, met);
02647
02648 /* Calculate geopotential heights... */
02649 read_met_geopot(ctl, met);
02650
02651 /* Calculate potential vorticity... */
02652 read_met_pv(met);
02653
02654 /* Calculate tropopause data... */
02655 read_met_tropo(ctl, met);
02656
02657 /* Calculate cloud properties... */
02658 read_met_cloud(met);
02659
02660 /* Calculate convective available potential energy... */
02661 read_met_cape(met);
02662
02663 /* Detrending... */
02664 read_met_detrend(ctl, met);
02665
02666 /* Close file... */
02667 NC(nc_close(ncid));
02668
02669 /* Return success... */
02670 return 1;
02671 }
02672
02673 /*****
02674
02675 void read_met_cape(
02676     met_t * met) {
02677
02678     /* Set timer... */
02679     SELECT_TIMER("READ_MET_CAPE", NVTX_READ);
02680
02681     /* Vertical spacing (about 100 m)... */
02682     const double pfac = 1.01439, dz0 = RI / MA / G0 * log(pfac);
02683
02684     /* Loop over columns... */
02685     #pragma omp parallel for default(shared)
02686     for (int ix = 0; ix < met->nx; ix++)
02687         for (int iy = 0; iy < met->ny; iy++) {
02688
02689             /* Get potential temperature and water vapor vmr at lowest 50 hPa... */
02690             int n = 0;
02691             double h2o = 0, t, theta = 0;
02692             double pbot = GSL_MIN(met->ps[ix][iy], met->p[0]);
02693             double ptop = pbot - 50.;
02694             for (int ip = 0; ip < met->np; ip++) {
02695                 if (met->p[ip] <= pbot) {
02696                     theta += THETA(met->p[ip], met->t[ix][iy][ip]);
02697                     h2o += met->h2o[ix][iy][ip];
02698                     n++;
02699                 }
02700                 if (met->p[ip] < ptop && n > 0)
02701                     break;
02702             }
02703             theta /= n;
02704             h2o /= n;
02705
02706             /* Cannot compute anything if water vapor is missing... */
02707             met->plcl[ix][iy] = GSL_NAN;
02708             met->plfc[ix][iy] = GSL_NAN;
02709             met->pel[ix][iy] = GSL_NAN;
02710             met->cape[ix][iy] = GSL_NAN;
02711             if (h2o <= 0)
02712                 continue;
02713

```



```

02714      /* Find lifted condensation level (LCL)... */
02715      ptop = P(20.);
02716      pbot = met->ps[ix][iy];
02717      do {
02718          met->plcl[ix][iy] = (float) (0.5 * (pbot + ptop));
02719          t = theta / pow(1000. / met->plcl[ix][iy], 0.286);
02720          if (RH(met->plcl[ix][iy], t, h2o) > 100.)
02721              ptop = met->plcl[ix][iy];
02722          else
02723              pbot = met->plcl[ix][iy];
02724      } while (pbot - ptop > 0.1);
02725
02726      /* Calculate level of free convection (LFC), equilibrium level (EL),
02727      and convective available potential energy (CAPE)... */
02728      double dcape = 0, dcape_old, dz, h2o_env, p = met->plcl[ix][iy],
02729      psat, t_env;
02730      ptop = 0.75 * clim_tropo(met->time, met->lat[iy]);
02731      met->cape[ix][iy] = 0;
02732      do {
02733          dz = dz0 * TVIRT(t, h2o);
02734          p /= pfac;
02735          t -= lapse_rate(t, h2o) * dz;
02736          psat = PSAT(t);
02737          h2o = psat / (p - (1. - EPS) * psat);
02738          INTPOL_INIT;
02739          intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
02740              &t_env, ci, cw, 1);
02741          intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
02742              &h2o_env, ci, cw, 0);
02743          dcape_old = dcape;
02744          dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
02745              TVIRT(t_env, h2o_env) * dz;
02746          if (dcape > 0) {
02747              met->cape[ix][iy] += (float) dcape;
02748              if (!isfinite(met->plfc[ix][iy]))
02749                  met->plfc[ix][iy] = (float) p;
02750          } else if (dcape_old > 0)
02751              met->pel[ix][iy] = (float) p;
02752      } while (p > ptop);
02753  }
02754 }
02755
02756 /*****
02757
02758 void read_met_cloud(
02759     met_t * met) {
02760
02761     /* Set timer... */
02762     SELECT_TIMER("READ_MET_CLOUD", NVTX_READ);
02763
02764     /* Loop over columns... */
02765     #pragma omp parallel for default(shared)
02766     for (int ix = 0; ix < met->nx; ix++)
02767         for (int iy = 0; iy < met->ny; iy++) {
02768
02769             /* Init... */
02770             met->pc[ix][iy] = GSL_NAN;
02771             met->cl[ix][iy] = 0;
02772
02773             /* Loop over pressure levels... */
02774             for (int ip = 0; ip < met->np - 1; ip++) {
02775
02776                 /* Check pressure... */
02777                 if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
02778                     continue;
02779
02780                 /* Get cloud top pressure ... */
02781                 if (met->iwc[ix][iy][ip] > 0 || met->lwc[ix][iy][ip] > 0)
02782                     met->pc[ix][iy] = (float) met->p[ip + 1];
02783
02784                 /* Get cloud water... */
02785                 met->cl[ix][iy] += (float)
02786                     (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1]
02787                         + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
02788                     * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
02789             }
02790         }
02791     }
02792
02793 /*****
02794
02795 void read_met_detrend(
02796     ctl_t * ctl,
02797     met_t * met) {
02798
02799     met_t *help;
02800

```

```

02801  /* Check parameters... */
02802  if (ctl->met_detrend <= 0)
02803      return;
02804
02805  /* Set timer... */
02806  SELECT_TIMER("READ_MET_DETREND", NVTX_READ);
02807
02808  /* Allocate... */
02809  ALLOC(help, met_t, 1);
02810
02811  /* Calculate standard deviation... */
02812  double sigma = ctl->met_detrend / 2.355;
02813  double tssq = 2. * SQR(sigma);
02814
02815  /* Calculate box size in latitude... */
02816  int sy = (int) (3. * DY2DEG(sigma) / fabs(met->lat[1] - met->lat[0]));
02817  sy = GSL_MIN(GSL_MAX(1, sy), met->ny / 2);
02818
02819  /* Calculate background... */
02820  #pragma omp parallel for default(shared)
02821  for (int ix = 0; ix < met->nx; ix++) {
02822      for (int iy = 0; iy < met->ny; iy++) {
02823
02824          /* Calculate Cartesian coordinates... */
02825          double x0[3];
02826          geo2cart(0.0, met->lon[ix], met->lat[iy], x0);
02827
02828          /* Calculate box size in longitude... */
02829          int sx =
02830              (int) (3. * DX2DEG(sigma, met->lat[iy]) /
02831                  fabs(met->lon[1] - met->lon[0]));
02832          sx = GSL_MIN(GSL_MAX(1, sx), met->nx / 2);
02833
02834          /* Init... */
02835          float wsum = 0;
02836          for (int ip = 0; ip < met->np; ip++) {
02837              help->t[ix][iy][ip] = 0;
02838              help->u[ix][iy][ip] = 0;
02839              help->v[ix][iy][ip] = 0;
02840              help->w[ix][iy][ip] = 0;
02841          }
02842
02843          /* Loop over neighboring grid points... */
02844          for (int ix2 = ix - sx; ix2 <= ix + sx; ix2++) {
02845              int ix3 = ix2;
02846              if (ix3 < 0)
02847                  ix3 += met->nx;
02848              else if (ix3 >= met->nx)
02849                  ix3 -= met->nx;
02850              for (int iy2 = GSL_MAX(iy - sy, 0);
02851                  iy2 <= GSL_MIN(iy + sy, met->ny - 1); iy2++) {
02852
02853                  /* Calculate Cartesian coordinates... */
02854                  double x1[3];
02855                  geo2cart(0.0, met->lon[ix3], met->lat[iy2], x1);
02856
02857                  /* Calculate weighting factor... */
02858                  float w = (float) exp(-DIST2(x0, x1) / tssq);
02859
02860                  /* Add data... */
02861                  wsum += w;
02862                  for (int ip = 0; ip < met->np; ip++) {
02863                      help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip];
02864                      help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip];
02865                      help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip];
02866                      help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip];
02867                  }
02868              }
02869          }
02870
02871          /* Normalize... */
02872          for (int ip = 0; ip < met->np; ip++) {
02873              help->t[ix][iy][ip] /= wsum;
02874              help->u[ix][iy][ip] /= wsum;
02875              help->v[ix][iy][ip] /= wsum;
02876              help->w[ix][iy][ip] /= wsum;
02877          }
02878      }
02879  }
02880
02881  /* Subtract background... */
02882  #pragma omp parallel for default(shared)
02883  for (int ix = 0; ix < met->nx; ix++)
02884      for (int iy = 0; iy < met->ny; iy++)
02885          for (int ip = 0; ip < met->np; ip++) {
02886              met->t[ix][iy][ip] -= help->t[ix][iy][ip];
02887              met->u[ix][iy][ip] -= help->u[ix][iy][ip];

```

```

02888         met->v[ix][iy][ip] -= help->v[ix][iy][ip];
02889         met->w[ix][iy][ip] -= help->w[ix][iy][ip];
02890     }
02891
02892     /* Free... */
02893     free(help);
02894 }
02895
02896 /*****
02897
02898 void read_met_extrapolate(
02899     met_t * met) {
02900
02901     int ip, ip0, ix, iy;
02902
02903     /* Set timer... */
02904     SELECT_TIMER("READ_MET_EXTRAPOLATE", NVTX_READ);
02905
02906     /* Loop over columns... */
02907     #pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
02908     for (ix = 0; ix < met->nx; ix++)
02909         for (iy = 0; iy < met->ny; iy++) {
02910
02911             /* Find lowest valid data point... */
02912             for (ip0 = met->np - 1; ip0 >= 0; ip0--)
02913                 if (!isfinite(met->t[ix][iy][ip0])
02914                     || !isfinite(met->u[ix][iy][ip0])
02915                     || !isfinite(met->v[ix][iy][ip0])
02916                     || !isfinite(met->w[ix][iy][ip0]))
02917                     break;
02918
02919             /* Extrapolate... */
02920             for (ip = ip0; ip >= 0; ip--) {
02921                 met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
02922                 met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
02923                 met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
02924                 met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
02925                 met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
02926                 met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
02927                 met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
02928                 met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
02929             }
02930         }
02931 }
02932
02933 /*****
02934
02935 void read_met_geopot(
02936     ctl_t * ctl,
02937     met_t * met) {
02938
02939     const int dx = ctl->met_geopot_sx, dy = ctl->met_geopot_sy;
02940
02941     static float help[EX][EY][EP], w, wsum;
02942
02943     double h2os, logp[EP], ts, z0;
02944
02945     int ip, ip0, ix, ix2, ix3, iy, iy2;
02946
02947     /* Set timer... */
02948     SELECT_TIMER("READ_MET_GEOPOT", NVTX_READ);
02949
02950     /* Calculate log pressure... */
02951     for (ip = 0; ip < met->np; ip++)
02952         logp[ip] = log(met->p[ip]);
02953
02954     /* Initialize geopotential heights... */
02955     #pragma omp parallel for default(shared) private(ix,iy,ip)
02956     for (ix = 0; ix < met->nx; ix++)
02957         for (iy = 0; iy < met->ny; iy++)
02958             for (ip = 0; ip < met->np; ip++)
02959                 met->z[ix][iy][ip] = GSL_NAN;
02960
02961     /* Apply hydrostatic equation to calculate geopotential heights... */
02962     #pragma omp parallel for default(shared) private(ix,iy,z0,ip0,ts,ip)
02963     for (ix = 0; ix < met->nx; ix++)
02964         for (iy = 0; iy < met->ny; iy++) {
02965
02966             /* Get surface height... */
02967             INTPOL_INIT;
02968             intpol_met_space_2d(met, met->z, met->lon[ix], met->lat[iy], &z0, ci,
02969                               cw, 1);
02970
02971             /* Find surface pressure level index... */
02972             ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
02973
02974             /* Get temperature and water vapor vmr at the surface... */

```

```

02975     ts =
02976         LIN(met->p[ip0], met->t[ix][iy][ip0], met->p[ip0 + 1],
02977             met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
02978     h2os =
02979         LIN(met->p[ip0], met->h2o[ix][iy][ip0], met->p[ip0 + 1],
02980             met->h2o[ix][iy][ip0 + 1], met->ps[ix][iy]);
02981
02982     /* Upper part of profile... */
02983     met->z[ix][iy][ip0 + 1]
02984         = (float) (z0 +
02985             ZDIFF(log(met->ps[ix][iy]), ts, h2os, logp[ip0 + 1],
02986                 met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]));
02987     for (ip = ip0 + 2; ip < met->np; ip++)
02988         met->z[ix][iy][ip]
02989             = (float) (met->z[ix][iy][ip - 1] +
02990                 ZDIFF(logp[ip - 1], met->t[ix][iy][ip - 1],
02991                     met->h2o[ix][iy][ip - 1], logp[ip],
02992                     met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
02993
02994     /* Lower part of profile... */
02995     met->z[ix][iy][ip0]
02996         = (float) (z0 +
02997             ZDIFF(log(met->ps[ix][iy]), ts, h2os, logp[ip0],
02998                 met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]));
02999     for (ip = ip0 - 1; ip >= 0; ip--)
03000         met->z[ix][iy][ip]
03001             = (float) (met->z[ix][iy][ip + 1] +
03002                 ZDIFF(logp[ip + 1], met->t[ix][iy][ip + 1],
03003                     met->h2o[ix][iy][ip + 1], logp[ip],
03004                     met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
03005 }
03006
03007 /* Horizontal smoothing... */
03008 #pragma omp parallel for default(shared) private(ix,iy,ip,ix2,ix3,iy2,w,wsum)
03009 for (ix = 0; ix < met->nx; ix++)
03010     for (iy = 0; iy < met->ny; iy++)
03011         for (ip = 0; ip < met->np; ip++) {
03012             wsum = 0;
03013             help[ix][iy][ip] = 0;
03014             for (ix2 = ix - dx + 1; ix2 <= ix + dx - 1; ix2++) {
03015                 ix3 = ix2;
03016                 if (ix3 < 0)
03017                     ix3 += met->nx;
03018                 else if (ix3 >= met->nx)
03019                     ix3 -= met->nx;
03020                 for (iy2 = GSL_MAX(iy - dy + 1, 0);
03021                     iy2 <= GSL_MIN(iy + dy - 1, met->ny - 1); iy2++)
03022                     if (isfinite(met->z[ix3][iy2][ip])) {
03023                         w = (1.0f - (float) abs(ix - ix2) / (float) dx)
03024                             * (1.0f - (float) abs(iy - iy2) / (float) dy);
03025                         help[ix][iy][ip] += w * met->z[ix3][iy2][ip];
03026                         wsum += w;
03027                     }
03028             }
03029             if (wsum > 0)
03030                 help[ix][iy][ip] /= wsum;
03031             else
03032                 help[ix][iy][ip] = GSL_NAN;
03033         }
03034
03035 /* Copy data... */
03036 #pragma omp parallel for default(shared) private(ix,iy,ip)
03037 for (ix = 0; ix < met->nx; ix++)
03038     for (iy = 0; iy < met->ny; iy++)
03039         for (ip = 0; ip < met->np; ip++)
03040             met->z[ix][iy][ip] = help[ix][iy][ip];
03041 }
03042
03043 /*****
03044
03045 void read_met_grid(
03046     char *filename,
03047     int ncid,
03048     ctl_t *ctl,
03049     met_t *met) {
03050
03051     char levname[LEN], tstr[10];
03052
03053     int dimid, ip, varid, year, mon, day, hour;
03054
03055     size_t np, nx, ny;
03056
03057     /* Set timer... */
03058     SELECT_TIMER("READ_MET_GRID", NVTX_READ);
03059
03060     /* Get time from filename... */
03061     sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);

```

```

03062 year = atoi(tstr);
03063 sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
03064 mon = atoi(tstr);
03065 sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
03066 day = atoi(tstr);
03067 sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
03068 hour = atoi(tstr);
03069 time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
03070
03071 /* Get grid dimensions... */
03072 NC(nc_inq_dimid(ncid, "lon", &dimid));
03073 NC(nc_inq_dimlen(ncid, dimid, &nx));
03074 if (nx < 2 || nx > EX)
03075     ERRMSG("Number of longitudes out of range!");
03076
03077 NC(nc_inq_dimid(ncid, "lat", &dimid));
03078 NC(nc_inq_dimlen(ncid, dimid, &ny));
03079 if (ny < 2 || ny > EY)
03080     ERRMSG("Number of latitudes out of range!");
03081
03082 sprintf(levname, "lev");
03083 NC(nc_inq_dimid(ncid, levname, &dimid));
03084 NC(nc_inq_dimlen(ncid, dimid, &np));
03085 if (np == 1) {
03086     sprintf(levname, "lev_2");
03087     if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
03088         sprintf(levname, "plev");
03089         nc_inq_dimid(ncid, levname, &dimid);
03090     }
03091     NC(nc_inq_dimlen(ncid, dimid, &np));
03092 }
03093 if (np < 2 || np > EP)
03094     ERRMSG("Number of levels out of range!");
03095
03096 /* Store dimensions... */
03097 met->np = (int) np;
03098 met->nx = (int) nx;
03099 met->ny = (int) ny;
03100
03101 /* Read longitudes and latitudes... */
03102 NC(nc_inq_varid(ncid, "lon", &varid));
03103 NC(nc_get_var_double(ncid, varid, met->lon));
03104 NC(nc_inq_varid(ncid, "lat", &varid));
03105 NC(nc_get_var_double(ncid, varid, met->lat));
03106
03107 /* Read pressure levels... */
03108 if (ctl->met_np <= 0) {
03109     NC(nc_inq_varid(ncid, levname, &varid));
03110     NC(nc_get_var_double(ncid, varid, met->p));
03111     for (ip = 0; ip < met->np; ip++)
03112         met->p[ip] /= 100.;
03113 }
03114
03115 /* Set pressure levels... */
03116 else {
03117     met->np = ctl->met_np;
03118     for (ip = 0; ip < met->np; ip++)
03119         met->p[ip] = ctl->met_p[ip];
03120 }
03121
03122 /* Check ordering of pressure levels... */
03123 for (ip = 1; ip < met->np; ip++)
03124     if (met->p[ip - 1] < met->p[ip])
03125         ERRMSG("Pressure levels must be descending!");
03126 }
03127
03128 /*****
03129
03130 int read_met_help_3d(
03131     int ncid,
03132     char *varname,
03133     char *varname2,
03134     met_t *met,
03135     float dest[EX][EY][EP],
03136     float scl) {
03137
03138     float *help;
03139
03140     int ip, ix, iy, varid;
03141
03142     /* Check if variable exists... */
03143     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03144         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
03145             return 0;
03146
03147     /* Allocate... */
03148     ALLOC(help, float,

```

```

03149         EX * EY * EP);
03150
03151     /* Read data... */
03152     NC(nc_get_var_float(ncid, varid, help));
03153
03154     /* Copy and check data... */
03155     #pragma omp parallel for default(shared) private(ix,iy,ip)
03156     for (ix = 0; ix < met->nx; ix++)
03157         for (iy = 0; iy < met->ny; iy++)
03158             for (ip = 0; ip < met->np; ip++) {
03159                 dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
03160                 if (fabsf(dest[ix][iy][ip]) < 1e14f)
03161                     dest[ix][iy][ip] *= scl;
03162                 else
03163                     dest[ix][iy][ip] = GSL_NAN;
03164             }
03165
03166     /* Free... */
03167     free(help);
03168
03169     /* Return... */
03170     return 1;
03171 }
03172
03173 /*****
03174
03175 int read_met_help_2d(
03176     int ncid,
03177     char *varname,
03178     char *varname2,
03179     met_t * met,
03180     float dest[EX][EY],
03181     float scl) {
03182
03183     float *help;
03184
03185     int ix, iy, varid;
03186
03187     /* Check if variable exists... */
03188     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03189         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
03190             return 0;
03191
03192     /* Allocate... */
03193     ALLOC(help, float,
03194           EX * EY);
03195
03196     /* Read data... */
03197     NC(nc_get_var_float(ncid, varid, help));
03198
03199     /* Copy and check data... */
03200     #pragma omp parallel for default(shared) private(ix,iy)
03201     for (ix = 0; ix < met->nx; ix++)
03202         for (iy = 0; iy < met->ny; iy++) {
03203             dest[ix][iy] = help[iy * met->nx + ix];
03204             if (fabsf(dest[ix][iy]) < 1e14f)
03205                 dest[ix][iy] *= scl;
03206             else
03207                 dest[ix][iy] = GSL_NAN;
03208         }
03209
03210     /* Free... */
03211     free(help);
03212
03213     /* Return... */
03214     return 1;
03215 }
03216
03217 /*****
03218
03219 void read_met_levels(
03220     int ncid,
03221     ctl_t * ctl,
03222     met_t * met) {
03223
03224     /* Set timer... */
03225     SELECT_TIMER("READ_MET_LEVELS", NVTX_READ);
03226
03227     /* Read meteorological data... */
03228     if (!read_met_help_3d(ncid, "t", "T", met, met->t, 1.0))
03229         ERRMSG("Cannot read temperature!");
03230     if (!read_met_help_3d(ncid, "u", "U", met, met->u, 1.0))
03231         ERRMSG("Cannot read zonal wind!");
03232     if (!read_met_help_3d(ncid, "v", "V", met, met->v, 1.0))
03233         ERRMSG("Cannot read meridional wind!");
03234     if (!read_met_help_3d(ncid, "w", "W", met, met->w, 0.01f))
03235         WARN("Cannot read vertical velocity");

```

```

03236 if (!read_met_help_3d(ncid, "q", "Q", met, met->h2o, (float) (MA / MH2O)))
03237     WARN("Cannot read specific humidity!");
03238 if (!read_met_help_3d(ncid, "o3", "O3", met, met->o3, (float) (MA / MO3)))
03239     WARN("Cannot read ozone data!");
03240 if (!read_met_help_3d(ncid, "clwc", "CLWC", met, met->lwc, 1.0))
03241     WARN("Cannot read cloud liquid water content!");
03242 if (!read_met_help_3d(ncid, "ciwc", "CIWC", met, met->iwc, 1.0))
03243     WARN("Cannot read cloud ice water content!");
03244
03245 /* Transfer from model levels to pressure levels... */
03246 if (ctl->met_np > 0) {
03247
03248     /* Read pressure on model levels... */
03249     if (!read_met_help_3d(ncid, "pl", "PL", met, met->pl, 0.01f))
03250         ERRMSG("Cannot read pressure on model levels!");
03251
03252     /* Vertical interpolation from model to pressure levels... */
03253     read_met_ml2pl(ctl, met, met->t);
03254     read_met_ml2pl(ctl, met, met->u);
03255     read_met_ml2pl(ctl, met, met->v);
03256     read_met_ml2pl(ctl, met, met->w);
03257     read_met_ml2pl(ctl, met, met->h2o);
03258     read_met_ml2pl(ctl, met, met->o3);
03259     read_met_ml2pl(ctl, met, met->lwc);
03260     read_met_ml2pl(ctl, met, met->iwc);
03261 }
03262 }
03263
03264 /*****
03265
03266 void read_met_ml2pl(
03267     ctl_t * ctl,
03268     met_t * met,
03269     float var[EX][EY][EP]) {
03270
03271     double aux[EP], p[EP], pt;
03272
03273     int ip, ip2, ix, iy;
03274
03275     /* Set timer... */
03276     SELECT_TIMER("READ_MET_ML2PL", NVTX_READ);
03277
03278     /* Loop over columns... */
03279     #pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
03280     for (ix = 0; ix < met->nx; ix++)
03281         for (iy = 0; iy < met->ny; iy++) {
03282
03283             /* Copy pressure profile... */
03284             for (ip = 0; ip < met->np; ip++)
03285                 p[ip] = met->pl[ix][iy][ip];
03286
03287             /* Interpolate... */
03288             for (ip = 0; ip < ctl->met_np; ip++) {
03289                 pt = ctl->met_p[ip];
03290                 if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
03291                     pt = p[0];
03292                 else if ((pt > p[met->np - 1] && p[1] > p[0])
03293                     || (pt < p[met->np - 1] && p[1] < p[0]))
03294                     pt = p[met->np - 1];
03295                 ip2 = locate_irr(p, met->np, pt);
03296                 aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
03297                     p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
03298             }
03299
03300             /* Copy data... */
03301             for (ip = 0; ip < ctl->met_np; ip++)
03302                 var[ix][iy][ip] = (float) aux[ip];
03303         }
03304 }
03305
03306 /*****
03307
03308 void read_met_periodic(
03309     met_t * met) {
03310
03311     /* Set timer... */
03312     SELECT_TIMER("READ_MET_PERIODIC", NVTX_READ);
03313
03314     /* Check longitudes... */
03315     if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
03316         + met->lon[1] - met->lon[0] - 360) < 0.01))
03317         return;
03318
03319     /* Increase longitude counter... */
03320     if ((++met->nx) > EX)
03321         ERRMSG("Cannot create periodic boundary conditions!");
03322

```

```

03323  /* Set longitude... */
03324  met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->lon[0];
03325
03326  /* Loop over latitudes and pressure levels... */
03327  #pragma omp parallel for default(shared)
03328  for (int iy = 0; iy < met->ny; iy++) {
03329      met->ps[met->nx - 1][iy] = met->ps[0][iy];
03330      met->zs[met->nx - 1][iy] = met->zs[0][iy];
03331      for (int ip = 0; ip < met->np; ip++) {
03332          met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
03333          met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
03334          met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
03335          met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
03336          met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
03337          met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
03338          met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
03339          met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
03340      }
03341  }
03342 }
03343
03344 /*****
03345
03346 void read_met_pv(
03347     met_t * met) {
03348
03349     double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
03350         dtdp, dudp, dvdp, latr, vort, pows[EP];
03351
03352     int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iyl;
03353
03354     /* Set timer... */
03355     SELECT_TIMER("READ_MET_PV", NVTX_READ);
03356
03357     /* Set powers... */
03358     for (ip = 0; ip < met->np; ip++)
03359         pows[ip] = pow(1000. / met->p[ip], 0.286);
03360
03361     /* Loop over grid points... */
03362     #pragma omp parallel for default(shared)
03363     private(ix,ix0,ix1,iy,iy0,iyl,latr,dx,dy,c0,c1,cr,vort,ip,ip0,ip1,dp0,dp1,denom,dtdx,dvdx,dtdy,dudy,dtdp,dudp,dvdp)
03364     for (ix = 0; ix < met->nx; ix++) {
03365
03366         /* Set indices... */
03367         ix0 = GSL_MAX(ix - 1, 0);
03368         ix1 = GSL_MIN(ix + 1, met->nx - 1);
03369
03370         /* Loop over grid points... */
03371         for (iy = 0; iy < met->ny; iy++) {
03372
03373             /* Set indices... */
03374             iy0 = GSL_MAX(iy - 1, 0);
03375             iyl = GSL_MIN(iy + 1, met->ny - 1);
03376
03377             /* Set auxiliary variables... */
03378             latr = 0.5 * (met->lat[iyl] + met->lat[iy0]);
03379             dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
03380             dy = 1000. * DEG2DY(met->lat[iyl] - met->lat[iy0]);
03381             c0 = cos(met->lat[iy0] / 180. * M_PI);
03382             c1 = cos(met->lat[iyl] / 180. * M_PI);
03383             cr = cos(latr / 180. * M_PI);
03384             vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
03385
03386             /* Loop over grid points... */
03387             for (ip = 0; ip < met->np; ip++) {
03388
03389                 /* Get gradients in longitude... */
03390                 dtdx = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
03391                 dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
03392
03393                 /* Get gradients in latitude... */
03394                 dtdy = (met->t[ix][iyl][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
03395                 dudy = (met->u[ix][iyl][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
03396
03397                 /* Set indices... */
03398                 ip0 = GSL_MAX(ip - 1, 0);
03399                 ip1 = GSL_MIN(ip + 1, met->np - 1);
03400
03401                 /* Get gradients in pressure... */
03402                 dp0 = 100. * (met->p[ip] - met->p[ip0]);
03403                 dp1 = 100. * (met->p[ip1] - met->p[ip]);
03404                 if (ip != ip0 && ip != ip1) {
03405                     denom = dp0 * dp1 * (dp0 + dp1);
03406                     dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
03407                         - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
03408                         + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
03409                         / denom;

```



```

03409         dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
03410                 - dp1 * dp1 * met->u[ix][iy][ip0]
03411                 + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
03412         / denom;
03413         dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
03414                 - dp1 * dp1 * met->v[ix][iy][ip0]
03415                 + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
03416         / denom;
03417     } else {
03418         denom = dp0 + dp1;
03419         dtdp =
03420             (met->t[ix][iy][ip1] * pows[ip1] -
03421              met->t[ix][iy][ip0] * pows[ip0]) / denom;
03422         dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
03423         dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
03424     }
03425
03426     /* Calculate PV... */
03427     met->pv[ix][iy][ip] = (float)
03428         (1e6 * G0 *
03429          (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
03430 }
03431 }
03432 }
03433
03434 /* Fix for polar regions... */
03435 #pragma omp parallel for default(shared) private(ix,ip)
03436 for (ix = 0; ix < met->nx; ix++)
03437     for (ip = 0; ip < met->np; ip++) {
03438         met->pv[ix][0][ip]
03439             = met->pv[ix][1][ip]
03440             = met->pv[ix][2][ip];
03441         met->pv[ix][met->ny - 1][ip]
03442             = met->pv[ix][met->ny - 2][ip]
03443             = met->pv[ix][met->ny - 3][ip];
03444     }
03445 }
03446
03447 /*****
03448
03449 void read_met_sample(
03450     ctl_t * ctl,
03451     met_t * met) {
03452
03453     met_t *help;
03454
03455     float w, wsum;
03456
03457     int ip, ip2, ix, ix2, ix3, iy, iy2;
03458
03459     /* Check parameters... */
03460     if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
03461         && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
03462         return;
03463
03464     /* Set timer... */
03465     SELECT_TIMER("READ_MET_SAMPLE", NVTX_READ);
03466
03467     /* Allocate... */
03468     ALLOC(help, met_t, 1);
03469
03470     /* Copy data... */
03471     help->nx = met->nx;
03472     help->ny = met->ny;
03473     help->np = met->np;
03474     memcpy(help->lon, met->lon, sizeof(met->lon));
03475     memcpy(help->lat, met->lat, sizeof(met->lat));
03476     memcpy(help->p, met->p, sizeof(met->p));
03477
03478     /* Smoothing... */
03479     for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
03480         for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
03481             for (ip = 0; ip < met->np; ip += ctl->met_dp) {
03482                 help->ps[ix][iy] = 0;
03483                 help->zs[ix][iy] = 0;
03484                 help->t[ix][iy][ip] = 0;
03485                 help->u[ix][iy][ip] = 0;
03486                 help->v[ix][iy][ip] = 0;
03487                 help->w[ix][iy][ip] = 0;
03488                 help->h2o[ix][iy][ip] = 0;
03489                 help->o3[ix][iy][ip] = 0;
03490                 help->lwc[ix][iy][ip] = 0;
03491                 help->iwc[ix][iy][ip] = 0;
03492                 wsum = 0;
03493                 for (ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1; ix2++) {
03494                     ix3 = ix2;
03495                     if (ix3 < 0)

```

```

03496         ix3 += met->nx;
03497     else if (ix3 >= met->nx)
03498         ix3 -= met->nx;
03499
03500     for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
03501          iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
03502     for (ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
03503          ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
03504         w = (1.0f - (float) abs(ix - ix2) / (float) ctl->met_sx)
03505             * (1.0f - (float) abs(iy - iy2) / (float) ctl->met_sy)
03506             * (1.0f - (float) abs(ip - ip2) / (float) ctl->met_sp);
03507         help->ps[ix][iy] += w * met->ps[ix3][iy2];
03508         help->zs[ix][iy] += w * met->zs[ix3][iy2];
03509         help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
03510         help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
03511         help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
03512         help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
03513         help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
03514         help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
03515         help->lwc[ix][iy][ip] += w * met->lwc[ix3][iy2][ip2];
03516         help->iwc[ix][iy][ip] += w * met->iwc[ix3][iy2][ip2];
03517         wsum += w;
03518     }
03519 }
03520 help->ps[ix][iy] /= wsum;
03521 help->zs[ix][iy] /= wsum;
03522 help->t[ix][iy][ip] /= wsum;
03523 help->u[ix][iy][ip] /= wsum;
03524 help->v[ix][iy][ip] /= wsum;
03525 help->w[ix][iy][ip] /= wsum;
03526 help->h2o[ix][iy][ip] /= wsum;
03527 help->o3[ix][iy][ip] /= wsum;
03528 help->lwc[ix][iy][ip] /= wsum;
03529 help->iwc[ix][iy][ip] /= wsum;
03530 }
03531 }
03532 }
03533
03534 /* Downsampling... */
03535 met->nx = 0;
03536 for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
03537     met->lon[met->nx] = help->lon[ix];
03538     met->ny = 0;
03539     for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
03540         met->lat[met->ny] = help->lat[iy];
03541         met->ps[met->nx][met->ny] = help->ps[ix][iy];
03542         met->zs[met->nx][met->ny] = help->zs[ix][iy];
03543         met->np = 0;
03544         for (ip = 0; ip < help->np; ip += ctl->met_dp) {
03545             met->p[met->np] = help->p[ip];
03546             met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
03547             met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
03548             met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
03549             met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
03550             met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
03551             met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
03552             met->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];
03553             met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
03554             met->np++;
03555         }
03556         met->ny++;
03557     }
03558     met->nx++;
03559 }
03560
03561 /* Free... */
03562 free(help);
03563 }
03564
03565 /*****
03566
03567 void read_met_surface(
03568     int ncid,
03569     met_t * met) {
03570
03571     int ix, iy;
03572
03573     /* Set timer... */
03574     SELECT_TIMER("READ_MET_SURFACE", NVTX_READ);
03575
03576     /* Read surface pressure... */
03577     if (!read_met_help_2d(ncid, "ps", "PS", met, met->ps, 0.01f)) {
03578         if (!read_met_help_2d(ncid, "lnsp", "LNSP", met, met->ps, 1.0)) {
03579             ERRMSG("Cannot not read surface pressure data!");
03580             for (ix = 0; ix < met->nx; ix++)
03581                 for (iy = 0; iy < met->ny; iy++)
03582                     met->ps[ix][iy] = (float) met->p[0];

```

```

03583     } else {
03584         for (iy = 0; iy < met->ny; iy++)
03585             for (ix = 0; ix < met->nx; ix++)
03586                 met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
03587     }
03588 }
03589
03590 /* Read geopotential height at the surface... */
03591 if (!read_met_help_2d
03592     (ncid, "z", "Z", met, met->zs, (float) (1. / (1000. * G0))))
03593     if (!read_met_help_2d
03594         (ncid, "zm", "ZM", met, met->zs, (float) (1. / 1000.)))
03595         ERRMSG("Cannot read surface geopotential height!");
03596
03597 /* Read temperature at the surface... */
03598 if (!read_met_help_2d(ncid, "t2m", "T2M", met, met->ts, 1.0))
03599     WARN("Cannot read surface temperature!");
03600
03601 /* Read zonal wind at the surface... */
03602 if (!read_met_help_2d(ncid, "u10m", "U10M", met, met->us, 1.0))
03603     WARN("Cannot read surface zonal wind!");
03604
03605 /* Read meridional wind at the surface... */
03606 if (!read_met_help_2d(ncid, "v10m", "V10M", met, met->vs, 1.0))
03607     WARN("Cannot read surface meridional wind!");
03608 }
03609
03610 /*****
03611 void read_met_tropo(
03612     ctl_t * ctl,
03613     met_t * met) {
03614
03615     double h2ot, p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
03616         th2[200], tt, z[EP], z2[200], zt;
03617
03618     int found, ix, iy, iz, iz2;
03619
03620     /* Set timer... */
03621     SELECT_TIMER("READ_MET_TROPO", NVTX_READ);
03622
03623     /* Get altitude and pressure profiles... */
03624     for (iz = 0; iz < met->np; iz++)
03625         z[iz] = Z(met->p[iz]);
03626     for (iz = 0; iz <= 190; iz++) {
03627         z2[iz] = 4.5 + 0.1 * iz;
03628         p2[iz] = P(z2[iz]);
03629     }
03630
03631     /* Do not calculate tropopause... */
03632     if (ctl->met_tropo == 0)
03633         for (ix = 0; ix < met->nx; ix++)
03634             for (iy = 0; iy < met->ny; iy++)
03635                 met->pt[ix][iy] = GSL_NAN;
03636
03637     /* Use tropopause climatology... */
03638     else if (ctl->met_tropo == 1) {
03639 #pragma omp parallel for default(shared) private(ix,iy)
03640         for (ix = 0; ix < met->nx; ix++)
03641             for (iy = 0; iy < met->ny; iy++)
03642                 met->pt[ix][iy] = (float) clim_tropo(met->time, met->lat[iy]);
03643     }
03644
03645     /* Use cold point... */
03646     else if (ctl->met_tropo == 2) {
03647
03648         /* Loop over grid points... */
03649 #pragma omp parallel for default(shared) private(ix,iy,iz,t,t2)
03650         for (ix = 0; ix < met->nx; ix++)
03651             for (iy = 0; iy < met->ny; iy++) {
03652
03653                 /* Interpolate temperature profile... */
03654                 for (iz = 0; iz < met->np; iz++)
03655                     t[iz] = met->t[ix][iy][iz];
03656                 spline(z, t, met->np, z2, t2, 171);
03657
03658                 /* Find minimum... */
03659                 iz = (int) gsl_stats_min_index(t2, 1, 171);
03660                 if (iz > 0 && iz < 170)
03661                     met->pt[ix][iy] = (float) p2[iz];
03662                 else
03663                     met->pt[ix][iy] = GSL_NAN;
03664             }
03665     }
03666
03667     /* Use WMO definition... */
03668     else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {

```

```

03670
03671  /* Loop over grid points... */
03672  #pragma omp parallel for default(shared) private(ix,iy,iz,iz2,t,t2,found)
03673  for (ix = 0; ix < met->nx; ix++)
03674  for (iy = 0; iy < met->ny; iy++) {
03675
03676      /* Interpolate temperature profile... */
03677      for (iz = 0; iz < met->np; iz++)
03678          t[iz] = met->t[ix][iy][iz];
03679      spline(z, t, met->np, z2, t2, 191);
03680
03681      /* Find 1st tropopause... */
03682      met->pt[ix][iy] = GSL_NAN;
03683      for (iz = 0; iz <= 170; iz++) {
03684          found = 1;
03685          for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
03686              if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) > 2.0) {
03687                  found = 0;
03688                  break;
03689              }
03690          if (found) {
03691              if (iz > 0 && iz < 170)
03692                  met->pt[ix][iy] = (float) p2[iz];
03693              break;
03694          }
03695      }
03696
03697      /* Find 2nd tropopause... */
03698      if (ctl->met_tropo == 4) {
03699          met->pt[ix][iy] = GSL_NAN;
03700          for (; iz <= 170; iz++) {
03701              found = 1;
03702              for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
03703                  if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) < 3.0) {
03704                      found = 0;
03705                      break;
03706                  }
03707              if (found)
03708                  break;
03709          }
03710          for (; iz <= 170; iz++) {
03711              found = 1;
03712              for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
03713                  if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) > 2.0) {
03714                      found = 0;
03715                      break;
03716                  }
03717              if (found) {
03718                  if (iz > 0 && iz < 170)
03719                      met->pt[ix][iy] = (float) p2[iz];
03720                  break;
03721              }
03722          }
03723      }
03724  }
03725  }
03726
03727  /* Use dynamical tropopause... */
03728  else if (ctl->met_tropo == 5) {
03729
03730      /* Loop over grid points... */
03731      #pragma omp parallel for default(shared) private(ix,iy,iz,pv,pv2,th,th2)
03732      for (ix = 0; ix < met->nx; ix++)
03733      for (iy = 0; iy < met->ny; iy++) {
03734
03735          /* Interpolate potential vorticity profile... */
03736          for (iz = 0; iz < met->np; iz++)
03737              pv[iz] = met->pv[ix][iy][iz];
03738          spline(z, pv, met->np, z2, pv2, 171);
03739
03740          /* Interpolate potential temperature profile... */
03741          for (iz = 0; iz < met->np; iz++)
03742              th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
03743          spline(z, th, met->np, z2, th2, 171);
03744
03745          /* Find dynamical tropopause 3.5 PVU + 380 K */
03746          met->pt[ix][iy] = GSL_NAN;
03747          for (iz = 0; iz <= 170; iz++)
03748              if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
03749                  if (iz > 0 && iz < 170)
03750                      met->pt[ix][iy] = (float) p2[iz];
03751                  break;
03752              }
03753      }
03754  }
03755  else

```

```

03757     ERRMSG("Cannot calculate tropopause!");
03758
03759     /* Interpolate temperature, geopotential height, and water vapor vmr... */
03760 #pragma omp parallel for default(shared) private(ix,iy,tt,zt,h2ot)
03761     for (ix = 0; ix < met->nx; ix++)
03762         for (iy = 0; iy < met->ny; iy++) {
03763             INTPOL_INIT;
03764             intpol_met_space_3d(met, met->t, met->pt[ix][iy], met->lon[ix],
03765                               met->lat[iy], &tt, ci, cw, 1);
03766             intpol_met_space_3d(met, met->z, met->pt[ix][iy], met->lon[ix],
03767                               met->lat[iy], &zt, ci, cw, 0);
03768             intpol_met_space_3d(met, met->h2o, met->pt[ix][iy], met->lon[ix],
03769                               met->lat[iy], &h2ot, ci, cw, 0);
03770             met->tt[ix][iy] = (float) tt;
03771             met->zt[ix][iy] = (float) zt;
03772             met->h2ot[ix][iy] = (float) h2ot;
03773         }
03774     }
03775
03776     /*****
03777
03778 double scan_ctl(
03779     const char *filename,
03780     int argc,
03781     char *argv[],
03782     const char *varname,
03783     int arridx,
03784     const char *defvalue,
03785     char *value) {
03786
03787     FILE *in = NULL;
03788
03789     char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
03790           msg[2 * LEN], rvarname[LEN], rval[LEN];
03791
03792     int contain = 0, i;
03793
03794     /* Open file... */
03795     if (filename[strlen(filename) - 1] != '-')
03796         if (!(in = fopen(filename, "r")))
03797             ERRMSG("Cannot open file!");
03798
03799     /* Set full variable name... */
03800     if (arridx >= 0) {
03801         sprintf(fullname1, "%s[%d]", varname, arridx);
03802         sprintf(fullname2, "%s[*]", varname);
03803     } else {
03804         sprintf(fullname1, "%s", varname);
03805         sprintf(fullname2, "%s", varname);
03806     }
03807
03808     /* Read data... */
03809     if (in != NULL)
03810         while (fgets(line, LEN, in))
03811             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
03812                 if (strcasecmp(rvarname, fullname1) == 0 ||
03813                     strcasecmp(rvarname, fullname2) == 0) {
03814                     contain = 1;
03815                     break;
03816                 }
03817     for (i = 1; i < argc - 1; i++)
03818         if (strcasecmp(argv[i], fullname1) == 0 ||
03819             strcasecmp(argv[i], fullname2) == 0) {
03820             sprintf(rval, "%s", argv[i + 1]);
03821             contain = 1;
03822             break;
03823         }
03824
03825     /* Close file... */
03826     if (in != NULL)
03827         fclose(in);
03828
03829     /* Check for missing variables... */
03830     if (!contain) {
03831         if (strlen(defvalue) > 0)
03832             sprintf(rval, "%s", defvalue);
03833         else {
03834             sprintf(msg, "Missing variable %s!\n", fullname1);
03835             ERRMSG(msg);
03836         }
03837     }
03838
03839     /* Write info... */
03840     printf("%s = %s\n", fullname1, rval);
03841
03842     /* Return values... */
03843     if (value != NULL)

```

```

03844     sprintf(value, "%s", rval);
03845     return atof(rval);
03846 }
03847
03848 /*****
03849
03850 double sedi(
03851     double p,
03852     double T,
03853     double r_p,
03854     double rho_p) {
03855
03856     double eta, G, K, lambda, rho, v;
03857
03858     /* Convert units... */
03859     p *= 100.;                      /* from hPa to Pa */
03860     r_p *= 1e-6;                    /* from microns to m */
03861
03862     /* Density of dry air... */
03863     rho = p / (RA * T);
03864
03865     /* Dynamic viscosity of air... */
03866     eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
03867
03868     /* Thermal velocity of an air molecule... */
03869     v = sqrt(8. * KB * T / (M_PI * 4.8096e-26));
03870
03871     /* Mean free path of an air molecule... */
03872     lambda = 2. * eta / (rho * v);
03873
03874     /* Knudsen number for air... */
03875     K = lambda / r_p;
03876
03877     /* Cunningham slip-flow correction... */
03878     G = 1. + K * (1.249 + 0.42 * exp(-0.87 / K));
03879
03880     /* Sedimentation velocity... */
03881     return 2. * SQR(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;
03882 }
03883
03884 /*****
03885
03886 void spline(
03887     double *x,
03888     double *y,
03889     int n,
03890     double *x2,
03891     double *y2,
03892     int n2) {
03893
03894     gsl_interp_accel *acc;
03895
03896     gsl_spline *s;
03897
03898     /* Allocate... */
03899     acc = gsl_interp_accel_alloc();
03900     s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
03901
03902     /* Interpolate profile... */
03903     gsl_spline_init(s, x, y, (size_t) n);
03904     for (int i = 0; i < n2; i++)
03905         if (x2[i] <= x[0])
03906             y2[i] = y[0];
03907         else if (x2[i] >= x[n - 1])
03908             y2[i] = y[n - 1];
03909         else
03910             y2[i] = gsl_spline_eval(s, x2[i], acc);
03911
03912     /* Free... */
03913     gsl_spline_free(s);
03914     gsl_interp_accel_free(acc);
03915 }
03916
03917 /*****
03918
03919 double stddev(
03920     double *data,
03921     int n) {
03922
03923     if (n <= 0)
03924         return 0;
03925
03926     double avg = 0, rms = 0;
03927
03928     for (int i = 0; i < n; ++i)
03929         avg += data[i];
03930     avg /= n;

```

```

03931
03932     for (int i = 0; i < n; ++i)
03933         rms += SQR(data[i] - avg);
03934
03935     return sqrt(rms / (n - 1));
03936 }
03937
03938 /*****
03939
03940 void time2jsec(
03941     int year,
03942     int mon,
03943     int day,
03944     int hour,
03945     int min,
03946     int sec,
03947     double remain,
03948     double *jsec) {
03949
03950     struct tm t0, t1;
03951
03952     t0.tm_year = 100;
03953     t0.tm_mon = 0;
03954     t0.tm_mday = 1;
03955     t0.tm_hour = 0;
03956     t0.tm_min = 0;
03957     t0.tm_sec = 0;
03958
03959     t1.tm_year = year - 1900;
03960     t1.tm_mon = mon - 1;
03961     t1.tm_mday = day;
03962     t1.tm_hour = hour;
03963     t1.tm_min = min;
03964     t1.tm_sec = sec;
03965
03966     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
03967 }
03968
03969 /*****
03970
03971 void timer(
03972     const char *name,
03973     int output) {
03974
03975     static char namelist[NTIMER][100];
03976
03977     static double runtime[NTIMER], t0, t1;
03978
03979     static int it = -1, nt;
03980
03981     /* Get time... */
03982     t1 = omp_get_wtime();
03983
03984     /* Add elapsed timer to old timer... */
03985     if (it >= 0)
03986         runtime[it] += t1 - t0;
03987
03988     /* Identify ID of new timer... */
03989     for (it = 0; it < nt; it++)
03990         if (strcmp(name, namelist[it]) == 0)
03991             break;
03992
03993     /* Check whether this is a new timer... */
03994     if (it >= nt) {
03995         sprintf(namelist[it], "%s", name);
03996         if ((++nt) > NTIMER)
03997             ERRMSG("Too many timers!");
03998     }
03999
04000     /* Save starting time... */
04001     t0 = t1;
04002
04003     /* Report timers... */
04004     if (output) {
04005         for (int it2 = 0; it2 < nt; it2++)
04006             printf("TIMER_%s = %.3f s\n", namelist[it2], runtime[it2]);
04007         double total = 0.0;
04008         for (int it2 = 0; it2 < nt; it2++)
04009             total += runtime[it2];
04010         printf("TIMER_TOTAL = %.3f s\n", total);
04011     }
04012 }
04013
04014 /*****
04015
04016 void write_atm(
04017     const char *filename,

```

```

04018     ctl_t * ctl,
04019     atm_t * atm,
04020     double t) {
04021
04022     FILE *in, *out;
04023
04024     char line[LEN];
04025
04026     double r, t0, t1;
04027
04028     int ip, iq, year, mon, day, hour, min, sec;
04029
04030     /* Set timer... */
04031     SELECT_TIMER("WRITE_ATM", NVTX_WRITE);
04032
04033     /* Set time interval for output... */
04034     t0 = t - 0.5 * ctl->dt_mod;
04035     t1 = t + 0.5 * ctl->dt_mod;
04036
04037     /* Write info... */
04038     printf("Write atmospheric data: %s\n", filename);
04039
04040     /* Write ASCII data... */
04041     if (ctl->atm_type == 0) {
04042
04043         /* Check if gnuplot output is requested... */
04044         if (ctl->atm_gpfile[0] != '-') {
04045
04046             /* Create gnuplot pipe... */
04047             if (!(out = popen("gnuplot", "w")))
04048                 ERRMSG("Cannot create pipe to gnuplot!");
04049
04050             /* Set plot filename... */
04051             fprintf(out, "set out \"%s.png\"\n", filename);
04052
04053             /* Set time string... */
04054             jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
04055             fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
04056                     year, mon, day, hour, min);
04057
04058             /* Dump gnuplot file to pipe... */
04059             if (!(in = fopen(ctl->atm_gpfile, "r")))
04060                 ERRMSG("Cannot open file!");
04061             while (fgets(line, LEN, in))
04062                 fprintf(out, "%s", line);
04063             fclose(in);
04064         }
04065
04066         else {
04067
04068             /* Create file... */
04069             if (!(out = fopen(filename, "w")))
04070                 ERRMSG("Cannot create file!");
04071         }
04072
04073         /* Write header... */
04074         fprintf(out,
04075             "# $1 = time [s]\n"
04076             "# $2 = altitude [km]\n"
04077             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
04078         for (iq = 0; iq < ctl->nq; iq++)
04079             fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
04080                     ctl->qnt_unit[iq]);
04081         fprintf(out, "\n");
04082
04083         /* Write data... */
04084         for (ip = 0; ip < atm->np; ip += ctl->atm_stride) {
04085
04086             /* Check time... */
04087             if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
04088                 continue;
04089
04090             /* Write output... */
04091             fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
04092                     atm->lon[ip], atm->lat[ip]);
04093             for (iq = 0; iq < ctl->nq; iq++) {
04094                 fprintf(out, " ");
04095                 fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
04096             }
04097             fprintf(out, "\n");
04098         }
04099
04100         /* Close file... */
04101         fclose(out);
04102     }
04103
04104     /* Write binary data... */

```



```

04105     else if (ctl->atm_type == 1) {
04106
04107         /* Create file... */
04108         if (!(out = fopen(filename, "w")))
04109             ERRMSG("Cannot create file!");
04110
04111         /* Write data... */
04112         FWRITE(&atm->np, int,
04113             1,
04114             out);
04115         FWRITE(atm->time, double,
04116             (size_t) atm->np,
04117             out);
04118         FWRITE(atm->p, double,
04119             (size_t) atm->np,
04120             out);
04121         FWRITE(atm->lon, double,
04122             (size_t) atm->np,
04123             out);
04124         FWRITE(atm->lat, double,
04125             (size_t) atm->np,
04126             out);
04127         for (iq = 0; iq < ctl->nq; iq++)
04128             FWRITE(atm->q[iq], double,
04129                 (size_t) atm->np,
04130                 out);
04131
04132         /* Close file... */
04133         fclose(out);
04134     }
04135
04136     /* Error... */
04137     else
04138         ERRMSG("Atmospheric data type not supported!");
04139 }
04140
04141 /*****
04142
04143 void write_csi(
04144     const char *filename,
04145     ctl_t *ctl,
04146     atm_t *atm,
04147     double t) {
04148
04149     static FILE *in, *out;
04150
04151     static char line[LEN];
04152
04153     static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ], rt, rt_old = -1e99,
04154         rz, rlon, rlat, robs, t0, t1, area[GY], dlon, dlat, dz, lat,
04155         x[1000000], y[1000000], work[2000000];
04156
04157     static int obscount[GX][GY][GZ], ct, cx, cy, cz, ip, ix, iy, iz, n;
04158
04159     /* Set timer... */
04160     SELECT_TIMER("WRITE_CSI", NVTX_WRITE);
04161
04162     /* Init... */
04163     if (t == ctl->t_start) {
04164
04165         /* Check quantity index for mass... */
04166         if (ctl->qnt_m < 0)
04167             ERRMSG("Need quantity mass!");
04168
04169         /* Open observation data file... */
04170         printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
04171         if (!(in = fopen(ctl->csi_obsfile, "r")))
04172             ERRMSG("Cannot open file!");
04173
04174         /* Create new file... */
04175         printf("Write CSI data: %s\n", filename);
04176         if (!(out = fopen(filename, "w")))
04177             ERRMSG("Cannot create file!");
04178
04179         /* Write header... */
04180         fprintf(out,
04181             "\n# $1 = time [s]\n"
04182             "\n# $2 = number of hits (cx)\n"
04183             "\n# $3 = number of misses (cy)\n"
04184             "\n# $4 = number of false alarms (cz)\n"
04185             "\n# $5 = number of observations (cx + cy)\n"
04186             "\n# $6 = number of forecasts (cx + cz)\n"
04187             "\n# $7 = bias (ratio of forecasts and observations) [%]\n"
04188             "\n# $8 = probability of detection (POD) [%]\n"
04189             "\n# $9 = false alarm rate (FAR) [%]\n"
04190             "\n# $10 = critical success index (CSI) [%]\n");
04191         fprintf(out,

```

```

04192     "# $11 = hits associated with random chance\n"
04193     "# $12 = equitable threat score (ETS) [%%]\n"
04194     "# $13 = Pearson linear correlation coefficient\n"
04195     "# $14 = Spearman rank-order correlation coefficient\n"
04196     "# $15 = column density mean error (F - O) [kg/m^2]\n"
04197     "# $16 = column density root mean square error (RMSE) [kg/m^2]\n"
04198     "# $17 = column density mean absolute error [kg/m^2]\n"
04199     "# $18 = number of data points\n\n");
04200
04201     /* Set grid box size... */
04202     dz = (ctl->csi_z1 - ctl->csi_z0) / ctl->csi_nz;
04203     dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
04204     dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
04205
04206     /* Set horizontal coordinates... */
04207     for (iy = 0; iy < ctl->csi_ny; iy++) {
04208         lat = ctl->csi_lat0 + dlat * (iy + 0.5);
04209         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.) * cos(lat * M_PI / 180.);
04210     }
04211 }
04212
04213     /* Set time interval... */
04214     t0 = t - 0.5 * ctl->dt_mod;
04215     t1 = t + 0.5 * ctl->dt_mod;
04216
04217     /* Initialize grid cells... */
04218     #pragma omp parallel for default(shared) private(ix,iy,iz)
04219     for (ix = 0; ix < ctl->csi_nx; ix++)
04220         for (iy = 0; iy < ctl->csi_ny; iy++)
04221             for (iz = 0; iz < ctl->csi_nz; iz++)
04222                 modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
04223
04224     /* Read observation data... */
04225     while (fgets(line, LEN, in)) {
04226
04227         /* Read data... */
04228         if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &robs) !=
04229             5)
04230             continue;
04231
04232         /* Check time... */
04233         if (rt < t0)
04234             continue;
04235         if (rt > t1)
04236             break;
04237         if (rt < rt_old)
04238             ERRMSG("Time must be ascending!");
04239         rt_old = rt;
04240
04241         /* Check observation data... */
04242         if (!isfinite(robs))
04243             continue;
04244
04245         /* Calculate indices... */
04246         ix = (int) ((rln - ctl->csi_lon0) / dlon);
04247         iy = (int) ((rln - ctl->csi_lat0) / dlat);
04248         iz = (int) ((rz - ctl->csi_z0) / dz);
04249
04250         /* Check indices... */
04251         if (ix < 0 || ix >= ctl->csi_nx ||
04252             iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
04253             continue;
04254
04255         /* Get mean observation index... */
04256         obsmean[ix][iy][iz] += robs;
04257         obscount[ix][iy][iz]++;
04258     }
04259
04260     /* Analyze model data... */
04261     for (ip = 0; ip < atm->np; ip++) {
04262
04263         /* Check time... */
04264         if (atm->time[ip] < t0 || atm->time[ip] > t1)
04265             continue;
04266
04267         /* Get indices... */
04268         ix = (int) ((atm->lon[ip] - ctl->csi_lon0) / dlon);
04269         iy = (int) ((atm->lat[ip] - ctl->csi_lat0) / dlat);
04270         iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0) / dz);
04271
04272         /* Check indices... */
04273         if (ix < 0 || ix >= ctl->csi_nx ||
04274             iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
04275             continue;
04276
04277         /* Get total mass in grid cell... */
04278         modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];

```

```

04279     }
04280
04281     /* Analyze all grid cells... */
04282     for (ix = 0; ix < ctl->csi_nx; ix++)
04283         for (iy = 0; iy < ctl->csi_ny; iy++)
04284             for (iz = 0; iz < ctl->csi_nz; iz++) {
04285
04286                 /* Calculate mean observation index... */
04287                 if (obscount[ix][iy][iz] > 0)
04288                     obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
04289
04290                 /* Calculate column density... */
04291                 if (modmean[ix][iy][iz] > 0)
04292                     modmean[ix][iy][iz] /= (1e6 * area[iy]);
04293
04294                 /* Calculate CSI... */
04295                 if (obscount[ix][iy][iz] > 0) {
04296                     ct++;
04297                     if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
04298                         modmean[ix][iy][iz] >= ctl->csi_modmin)
04299                         cx++;
04300                     else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
04301                         modmean[ix][iy][iz] < ctl->csi_modmin)
04302                         cy++;
04303                     else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
04304                         modmean[ix][iy][iz] >= ctl->csi_modmin)
04305                         cz++;
04306                 }
04307
04308                 /* Save data for other verification statistics... */
04309                 if (obscount[ix][iy][iz] > 0
04310                     && (obsmean[ix][iy][iz] >= ctl->csi_obsmin
04311                         || modmean[ix][iy][iz] >= ctl->csi_modmin)) {
04312                     x[n] = modmean[ix][iy][iz];
04313                     y[n] = obsmean[ix][iy][iz];
04314                     if ((++n) > 1000000)
04315                         ERRMSG("Too many data points to calculate statistics!");
04316                 }
04317             }
04318
04319     /* Write output... */
04320     if (fmod(t, ctl->csi_dt_out) == 0) {
04321
04322         /* Calculate verification statistics
04323            (https://www.cawcr.gov.au/projects/verification/) ... */
04324         int nobs = cx + cy;
04325         int nfor = cx + cz;
04326         double bias = (nobs > 0) ? 100. * nfor / nobs : GSL_NAN;
04327         double pod = (nobs > 0) ? (100. * cx) / nobs : GSL_NAN;
04328         double far = (nfor > 0) ? (100. * cz) / nfor : GSL_NAN;
04329         double csi = (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN;
04330         double cx_rd = (ct > 0) ? (1. * nobs * nfor) / ct : GSL_NAN;
04331         double ets = (cx + cy + cz - cx_rd > 0) ?
04332             (100. * (cx - cx_rd)) / (cx + cy + cz - cx_rd) : GSL_NAN;
04333         double rho_p =
04334             (n > 0) ? gsl_stats_correlation(x, 1, y, 1, (size_t) n) : GSL_NAN;
04335         double rho_s =
04336             (n > 0) ? gsl_stats_spearman(x, 1, y, 1, (size_t) n, work) : GSL_NAN;
04337         for (int i = 0; i < n; i++)
04338             work[i] = x[i] - y[i];
04339         double mean = (n > 0) ? gsl_stats_mean(work, 1, (size_t) n) : GSL_NAN;
04340         double rmse = (n > 0) ? gsl_stats_sd_with_fixed_mean(work, 1, (size_t) n,
04341             0.0) : GSL_NAN;
04342         double absdev =
04343             (n > 0) ? gsl_stats_absdev_m(work, 1, (size_t) n, 0.0) : GSL_NAN;
04344
04345         /* Write... */
04346         fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g %g %g %g %g %d\n",
04347             t, cx, cy, cz, nobs, nfor, bias, pod, far, csi, cx_rd, ets,
04348             rho_p, rho_s, mean, rmse, absdev, n);
04349
04350         /* Set counters to zero... */
04351         n = ct = cx = cy = cz = 0;
04352     }
04353
04354     /* Close file... */
04355     if (t == ctl->t_stop)
04356         fclose(out);
04357 }
04358
04359 /*****
04360
04361 void write_ens(
04362     const char *filename,
04363     ctl_t * ctl,
04364     atm_t * atm,
04365     double t) {

```

```

04366
04367     static FILE *out;
04368
04369     static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
04370         t0, t1, x[NENS][3], xm[3];
04371
04372     static int ip, iq;
04373
04374     static size_t i, n;
04375
04376     /* Set timer... */
04377     SELECT_TIMER("WRITE_ENS", NVTX_WRITE);
04378
04379     /* Init... */
04380     if (t == ctl->t_start) {
04381
04382         /* Check quantities... */
04383         if (ctl->qnt_ens < 0)
04384             ERRMSG("Missing ensemble IDs!");
04385
04386         /* Create new file... */
04387         printf("Write ensemble data: %s\n", filename);
04388         if (!(out = fopen(filename, "w")))
04389             ERRMSG("Cannot create file!");
04390
04391         /* Write header... */
04392         fprintf(out,
04393             "# $1 = time [s]\n"
04394             "# $2 = altitude [km]\n"
04395             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
04396         for (iq = 0; iq < ctl->nq; iq++)
04397             fprintf(out, "# %d = %s (mean) [%s]\n", 5 + iq,
04398                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
04399         for (iq = 0; iq < ctl->nq; iq++)
04400             fprintf(out, "# %d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
04401                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
04402         fprintf(out, "# %d = number of members\n\n", 5 + 2 * ctl->nq);
04403     }
04404
04405     /* Set time interval... */
04406     t0 = t - 0.5 * ctl->dt_mod;
04407     t1 = t + 0.5 * ctl->dt_mod;
04408
04409     /* Init... */
04410     ens = GSL_NAN;
04411     n = 0;
04412
04413     /* Loop over air parcels... */
04414     for (ip = 0; ip < atm->np; ip++) {
04415
04416         /* Check time... */
04417         if (atm->time[ip] < t0 || atm->time[ip] > t1)
04418             continue;
04419
04420         /* Check ensemble id... */
04421         if (atm->q[ctl->qnt_ens][ip] != ens) {
04422
04423             /* Write results... */
04424             if (n > 0) {
04425
04426                 /* Get mean position... */
04427                 xm[0] = xm[1] = xm[2] = 0;
04428                 for (i = 0; i < n; i++) {
04429                     xm[0] += x[i][0] / (double) n;
04430                     xm[1] += x[i][1] / (double) n;
04431                     xm[2] += x[i][2] / (double) n;
04432                 }
04433                 cart2geo(xm, &dummy, &lon, &lat);
04434                 fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
04435                     lat);
04436
04437                 /* Get quantity statistics... */
04438                 for (iq = 0; iq < ctl->nq; iq++) {
04439                     fprintf(out, " ");
04440                     fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
04441                 }
04442                 for (iq = 0; iq < ctl->nq; iq++) {
04443                     fprintf(out, " ");
04444                     fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
04445                 }
04446                 fprintf(out, " %lu\n", n);
04447             }
04448
04449             /* Init new ensemble... */
04450             ens = atm->q[ctl->qnt_ens][ip];
04451             n = 0;
04452         }

```

```

04453
04454     /* Save data... */
04455     p[n] = atm->p[ip];
04456     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
04457     for (iq = 0; iq < ctl->nq; iq++)
04458         q[iq][n] = atm->q[iq][ip];
04459     if ((++n) >= NENS)
04460         ERRMSG("Too many data points!");
04461 }
04462
04463     /* Write results... */
04464     if (n > 0) {
04465
04466         /* Get mean position... */
04467         xm[0] = xm[1] = xm[2] = 0;
04468         for (i = 0; i < n; i++) {
04469             xm[0] += x[i][0] / (double) n;
04470             xm[1] += x[i][1] / (double) n;
04471             xm[2] += x[i][2] / (double) n;
04472         }
04473         cart2geo(xm, &dummy, &lon, &lat);
04474         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
04475
04476         /* Get quantity statistics... */
04477         for (iq = 0; iq < ctl->nq; iq++) {
04478             fprintf(out, " ");
04479             fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
04480         }
04481         for (iq = 0; iq < ctl->nq; iq++) {
04482             fprintf(out, " ");
04483             fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
04484         }
04485         fprintf(out, " %lu\n", n);
04486     }
04487
04488     /* Close file... */
04489     if (t == ctl->t_stop)
04490         fclose(out);
04491 }
04492
04493 /*****
04494
04495 void write_grid(
04496     const char *filename,
04497     ctl_t * ctl,
04498     met_t * met0,
04499     met_t * met1,
04500     atm_t * atm,
04501     double t) {
04502
04503     FILE *in, *out;
04504
04505     char line[LEN];
04506
04507     static double mass[GX][GY][GZ], z[GZ], dz, lon[GX], dlon, lat[GY], dlat,
04508         area[GY], rho_air, press[GZ], temp, cd, vmr, t0, t1, r;
04509
04510     static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec;
04511
04512     /* Set timer... */
04513     SELECT_TIMER("WRITE_GRID", NVTX_WRITE);
04514
04515     /* Check dimensions... */
04516     if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
04517         ERRMSG("Grid dimensions too large!");
04518
04519     /* Set grid box size... */
04520     dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
04521     dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
04522     dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
04523
04524     /* Set vertical coordinates... */
04525     for (iz = 0; iz < ctl->grid_nz; iz++) {
04526         z[iz] = ctl->grid_z0 + dz * (iz + 0.5);
04527         press[iz] = P(z[iz]);
04528     }
04529
04530     /* Set horizontal coordinates... */
04531     for (ix = 0; ix < ctl->grid_nx; ix++)
04532         lon[ix] = ctl->grid_lon0 + dlon * (ix + 0.5);
04533     for (iy = 0; iy < ctl->grid_ny; iy++) {
04534         lat[iy] = ctl->grid_lat0 + dlat * (iy + 0.5);
04535         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
04536             * cos(lat[iy] * M_PI / 180.);
04537     }
04538
04539     /* Set time interval for output... */

```

```

04540     t0 = t - 0.5 * ctl->dt_mod;
04541     t1 = t + 0.5 * ctl->dt_mod;
04542
04543     /* Initialize grid... */
04544 #pragma omp parallel for default(shared) private(ix,iy,iz)
04545     for (ix = 0; ix < ctl->grid_nx; ix++)
04546         for (iy = 0; iy < ctl->grid_ny; iy++)
04547             for (iz = 0; iz < ctl->grid_nz; iz++) {
04548                 mass[ix][iy][iz] = 0;
04549                 np[ix][iy][iz] = 0;
04550             }
04551
04552     /* Average data... */
04553     for (ip = 0; ip < atm->np; ip++)
04554         if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
04555
04556             /* Get index... */
04557             ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
04558             iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
04559             iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
04560
04561             /* Check indices... */
04562             if (ix < 0 || ix >= ctl->grid_nx ||
04563                 iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
04564                 continue;
04565
04566             /* Add mass... */
04567             if (ctl->qnt_m >= 0)
04568                 mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04569             np[ix][iy][iz]++;
04570         }
04571
04572     /* Check if gnuplot output is requested... */
04573     if (ctl->grid_gpfile[0] != '-') {
04574
04575         /* Write info... */
04576         printf("Plot grid data: %s.png\n", filename);
04577
04578         /* Create gnuplot pipe... */
04579         if (!(out = popen("gnuplot", "w")))
04580             ERRMSG("Cannot create pipe to gnuplot!");
04581
04582         /* Set plot filename... */
04583         fprintf(out, "set out \"%s.png\"\n", filename);
04584
04585         /* Set time string... */
04586         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
04587         fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
04588             year, mon, day, hour, min);
04589
04590         /* Dump gnuplot file to pipe... */
04591         if (!(in = fopen(ctl->grid_gpfile, "r")))
04592             ERRMSG("Cannot open file!");
04593         while (fgets(line, LEN, in))
04594             fprintf(out, "%s", line);
04595         fclose(in);
04596     }
04597
04598     else {
04599
04600         /* Write info... */
04601         printf("Write grid data: %s\n", filename);
04602
04603         /* Create file... */
04604         if (!(out = fopen(filename, "w")))
04605             ERRMSG("Cannot create file!");
04606     }
04607
04608     /* Write header... */
04609     fprintf(out,
04610         "# $1 = time [s]\n"
04611         "# $2 = altitude [km]\n"
04612         "# $3 = longitude [deg]\n"
04613         "# $4 = latitude [deg]\n"
04614         "# $5 = surface area [km^2]\n"
04615         "# $6 = layer width [km]\n"
04616         "# $7 = number of particles [l]\n"
04617         "# $8 = column density [kg/m^2]\n"
04618         "# $9 = volume mixing ratio [ppv]\n\n");
04619
04620     /* Write data... */
04621     for (ix = 0; ix < ctl->grid_nx; ix++) {
04622         if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
04623             fprintf(out, "\n");
04624         for (iy = 0; iy < ctl->grid_ny; iy++) {
04625             if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
04626                 fprintf(out, "\n");

```

```

04627     for (iz = 0; iz < ctl->grid_nz; iz++)
04628     if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
04629
04630         /* Calculate column density... */
04631         cd = mass[ix][iy][iz] / (1e6 * area[iy]);
04632
04633         /* Calculate volume mixing ratio... */
04634         vmr = 0;
04635         if (ctl->molmass > 0) {
04636             if (mass[ix][iy][iz] > 0) {
04637
04638                 /* Get temperature... */
04639                 INTPOL_INIT;
04640                 intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
04641                                 lon[ix], lat[iy], &temp, ci, cw, 1);
04642
04643                 /* Calculate density of air... */
04644                 rho_air = 100. * press[iz] / (RA * temp);
04645
04646                 /* Calculate volume mixing ratio... */
04647                 vmr = MA / ctl->molmass * mass[ix][iy][iz]
04648                     / (rho_air * 1e6 * area[iy] * 1e3 * dz);
04649             }
04650         } else
04651             vmr = GSL_NAN;
04652
04653         /* Write output... */
04654         fprintf(out, "%.2f %g %g %g %g %d %g %g\n", t, z[iz],
04655             lon[ix], lat[iy], area[iy], dz, np[ix][iy][iz], cd, vmr);
04656     }
04657 }
04658 }
04659
04660 /* Close file... */
04661 fclose(out);
04662 }
04663
04664 /*****
04665
04666 void write_prof(
04667     const char *filename,
04668     ctl_t * ctl,
04669     met_t * met0,
04670     met_t * met1,
04671     atm_t * atm,
04672     double t) {
04673
04674     static FILE *in, *out;
04675
04676     static char line[LEN];
04677
04678     static double mass[GX][GY][GZ], obsmean[GX][GY], rt, rt_old = -1e99,
04679         rz, rlon, rlat, robs, t0, t1, area[GY], dz, dlon, dlat, lon[GX], lat[GY],
04680         z[GZ], press[GZ], temp, rho_air, vmr, h2o, o3;
04681
04682     static int obscount[GX][GY], ip, ix, iy, iz, okay;
04683
04684     /* Set timer... */
04685     SELECT_TIMER("WRITE_PROF", NVTX_WRITE);
04686
04687     /* Init... */
04688     if (t == ctl->t_start) {
04689
04690         /* Check quantity index for mass... */
04691         if (ctl->qnt_m < 0)
04692             ERRMSG("Need quantity mass!");
04693
04694         /* Check dimensions... */
04695         if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
04696             ERRMSG("Grid dimensions too large!");
04697
04698         /* Check molar mass... */
04699         if (ctl->molmass <= 0)
04700             ERRMSG("Specify molar mass!");
04701
04702         /* Open observation data file... */
04703         printf("Read profile observation data: %s\n", ctl->prof_obsfile);
04704         if (!(in = fopen(ctl->prof_obsfile, "r")))
04705             ERRMSG("Cannot open file!");
04706
04707         /* Create new output file... */
04708         printf("Write profile data: %s\n", filename);
04709         if (!(out = fopen(filename, "w")))
04710             ERRMSG("Cannot create file!");
04711
04712         /* Write header... */
04713         fprintf(out,

```

```

04714         "# $1 = time [s]\n"
04715         "# $2 = altitude [km]\n"
04716         "# $3 = longitude [deg]\n"
04717         "# $4 = latitude [deg]\n"
04718         "# $5 = pressure [hPa]\n"
04719         "# $6 = temperature [K]\n"
04720         "# $7 = volume mixing ratio [ppv]\n"
04721         "# $8 = H2O volume mixing ratio [ppv]\n"
04722         "# $9 = O3 volume mixing ratio [ppv]\n"
04723         "# $10 = observed BT index [K]\n");
04724
04725     /* Set grid box size... */
04726     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
04727     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
04728     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
04729
04730     /* Set vertical coordinates... */
04731     for (iz = 0; iz < ctl->prof_nz; iz++) {
04732         z[iz] = ctl->prof_z0 + dz * (iz + 0.5);
04733         press[iz] = P(z[iz]);
04734     }
04735
04736     /* Set horizontal coordinates... */
04737     for (ix = 0; ix < ctl->prof_nx; ix++)
04738         lon[ix] = ctl->prof_lon0 + dlon * (ix + 0.5);
04739     for (iy = 0; iy < ctl->prof_ny; iy++) {
04740         lat[iy] = ctl->prof_lat0 + dlat * (iy + 0.5);
04741         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
04742             * cos(lat[iy] * M_PI / 180.);
04743     }
04744 }
04745
04746 /* Set time interval... */
04747 t0 = t - 0.5 * ctl->dt_mod;
04748 t1 = t + 0.5 * ctl->dt_mod;
04749
04750 /* Initialize... */
04751 #pragma omp parallel for default(shared) private(ix,iy,iz)
04752 for (ix = 0; ix < ctl->prof_nx; ix++)
04753     for (iy = 0; iy < ctl->prof_ny; iy++) {
04754         obsmean[ix][iy] = 0;
04755         obscount[ix][iy] = 0;
04756         for (iz = 0; iz < ctl->prof_nz; iz++)
04757             mass[ix][iy][iz] = 0;
04758     }
04759
04760 /* Read observation data... */
04761 while (fgets(line, LEN, in)) {
04762
04763     /* Read data... */
04764     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &robs) !=
04765         5)
04766         continue;
04767
04768     /* Check time... */
04769     if (rt < t0)
04770         continue;
04771     if (rt > t1)
04772         break;
04773     if (rt < rt_old)
04774         ERRMSG("Time must be ascending!");
04775     rt_old = rt;
04776
04777     /* Check observation data... */
04778     if (!isfinite(robs))
04779         continue;
04780
04781     /* Calculate indices... */
04782     ix = (int) ((rln - ctl->prof_lon0) / dlon);
04783     iy = (int) ((rln - ctl->prof_lat0) / dlat);
04784
04785     /* Check indices... */
04786     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
04787         continue;
04788
04789     /* Get mean observation index... */
04790     obsmean[ix][iy] += robs;
04791     obscount[ix][iy]++;
04792 }
04793
04794 /* Analyze model data... */
04795 for (ip = 0; ip < atm->np; ip++) {
04796
04797     /* Check time... */
04798     if (atm->time[ip] < t0 || atm->time[ip] > t1)
04799         continue;
04800

```



```

04801     /* Get indices... */
04802     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
04803     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
04804     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
04805
04806     /* Check indices... */
04807     if (ix < 0 || ix >= ctl->prof_nx ||
04808         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
04809         continue;
04810
04811     /* Get total mass in grid cell... */
04812     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04813 }
04814
04815 /* Extract profiles... */
04816 for (ix = 0; ix < ctl->prof_nx; ix++)
04817     for (iy = 0; iy < ctl->prof_ny; iy++)
04818         if (obscount[ix][iy] > 0) {
04819
04820             /* Check profile... */
04821             okay = 0;
04822             for (iz = 0; iz < ctl->prof_nz; iz++)
04823                 if (mass[ix][iy][iz] > 0) {
04824                     okay = 1;
04825                     break;
04826                 }
04827             if (!okay)
04828                 continue;
04829
04830             /* Write output... */
04831             fprintf(out, "\n");
04832
04833             /* Loop over altitudes... */
04834             for (iz = 0; iz < ctl->prof_nz; iz++) {
04835
04836                 /* Get pressure and temperature... */
04837                 INTPOL_INIT;
04838                 intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
04839                                     lon[ix], lat[iy], &temp, ci, cw, 1);
04840                 intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, t, press[iz],
04841                                     lon[ix], lat[iy], &h2o, ci, cw, 0);
04842                 intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press[iz],
04843                                     lon[ix], lat[iy], &o3, ci, cw, 0);
04844
04845                 /* Calculate volume mixing ratio... */
04846                 rho_air = 100. * press[iz] / (RA * temp);
04847                 vmr = MA / ctl->molmass * mass[ix][iy][iz]
04848                     / (rho_air * area[iy] * dz * 1e9);
04849
04850                 /* Write output... */
04851                 fprintf(out, "%.2f %g %g %g %g %g %g %g\n",
04852                         t, z[iz], lon[ix], lat[iy], press[iz], temp, vmr, h2o, o3,
04853                         obsmean[ix][iy] / obscount[ix][iy]);
04854             }
04855         }
04856
04857     /* Close files... */
04858     if (t == ctl->t_stop) {
04859         fclose(in);
04860         fclose(out);
04861     }
04862 }
04863
04864 /*****
04865
04866 void write_sample(
04867     const char *filename,
04868     ctl_t * ctl,
04869     met_t * met0,
04870     met_t * met1,
04871     atm_t * atm,
04872     double t) {
04873
04874     static FILE *in, *out;
04875
04876     static char line[LEN];
04877
04878     static double area, dlat, rmax2, t0, t1, rt, rt_old, rz, rlon, rlat, robs;
04879
04880     /* Set timer... */
04881     SELECT_TIMER("WRITE_SAMPLE", NVTX_WRITE);
04882
04883     /* Init... */
04884     if (t == ctl->t_start) {
04885
04886         /* Open observation data file... */
04887         printf("Read sample observation data: %s\n", ctl->sample_obsfile);

```

```

04888     if (!(in = fopen(ctl->sample_obsfile, "r")))
04889         ERRMSG("Cannot open file!");
04890
04891     /* Create new file... */
04892     printf("Write sample data: %s\n", filename);
04893     if (!(out = fopen(filename, "w")))
04894         ERRMSG("Cannot create file!");
04895
04896     /* Write header... */
04897     fprintf(out,
04898         "# $1 = time [s]\n"
04899         "# $2 = altitude [km]\n"
04900         "# $3 = longitude [deg]\n"
04901         "# $4 = latitude [deg]\n"
04902         "# $5 = surface area [km^2]\n"
04903         "# $6 = layer width [km]\n"
04904         "# $7 = number of particles [1]\n"
04905         "# $8 = column density [kg/m^2]\n"
04906         "# $9 = volume mixing ratio [ppv]\n"
04907         "# $10 = observed BT index [K]\n\n");
04908
04909     /* Set latitude range, squared radius, and area... */
04910     dlat = DY2DEG(ctl->sample_dx);
04911     rmax2 = SQR(ctl->sample_dx);
04912     area = M_PI * rmax2;
04913 }
04914
04915 /* Set time interval for output... */
04916 t0 = t - 0.5 * ctl->dt_mod;
04917 t1 = t + 0.5 * ctl->dt_mod;
04918
04919 /* Read observation data... */
04920 while (fgets(line, LEN, in)) {
04921
04922     /* Read data... */
04923     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rlat, &robs) !=
04924         5)
04925         continue;
04926
04927     /* Check time... */
04928     if (rt < t0)
04929         continue;
04930     if (rt < rt_old)
04931         ERRMSG("Time must be ascending!");
04932     rt_old = rt;
04933
04934     /* Calculate Cartesian coordinates... */
04935     double x0[3];
04936     geo2cart(0, rln, rlat, x0);
04937
04938     /* Set pressure range... */
04939     double rp = P(rz);
04940     double ptop = P(rz + ctl->sample_dz);
04941     double pbot = P(rz - ctl->sample_dz);
04942
04943     /* Init... */
04944     double mass = 0;
04945     int np = 0;
04946
04947     /* Loop over air parcels... */
04948 #pragma omp parallel for default(shared) reduction(+:mass,np)
04949     for (int ip = 0; ip < atm->np; ip++) {
04950
04951         /* Check time... */
04952         if (atm->time[ip] < t0 || atm->time[ip] > t1)
04953             continue;
04954
04955         /* Check latitude... */
04956         if (fabs(rlat - atm->lat[ip]) > dlat)
04957             continue;
04958
04959         /* Check horizontal distance... */
04960         double x1[3];
04961         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
04962         if (DIST2(x0, x1) > rmax2)
04963             continue;
04964
04965         /* Check pressure... */
04966         if (ctl->sample_dz > 0)
04967             if (atm->p[ip] > pbot || atm->p[ip] < ptop)
04968                 continue;
04969
04970         /* Add mass... */
04971         if (ctl->qnt_m >= 0)
04972             mass += atm->q[ctl->qnt_m][ip];
04973         np++;
04974     }

```

```

04975
04976 /* Calculate column density... */
04977 double cd = mass / (1e6 * area);
04978
04979 /* Calculate volume mixing ratio... */
04980 double vmr = 0;
04981 if (ctl->molmass > 0 && ctl->sample_dz > 0) {
04982     if (mass > 0) {
04983
04984         /* Get temperature... */
04985         double temp;
04986         INTPOL_INIT;
04987         intpol_met_time_3d(met0, met0->t, met1, met1->t, rt, rp,
04988             rlon, rlat, &temp, ci, cw, 1);
04989
04990         /* Calculate density of air... */
04991         double rho_air = 100. * rp / (RA * temp);
04992
04993         /* Calculate volume mixing ratio... */
04994         vmr = MA / ctl->molmass * mass
04995             / (rho_air * 1e6 * area * 1e3 * ctl->sample_dz);
04996     }
04997 } else
04998     vmr = GSL_NAN;
04999
05000 /* Write output... */
05001 fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n", rt, rz, rlon, rlat,
05002     area, ctl->sample_dz, np, cd, vmr, robs);
05003
05004 /* Check time... */
05005 if (rt >= t1)
05006     break;
05007 }
05008
05009 /* Close files... */
05010 if (t == ctl->t_stop) {
05011     fclose(in);
05012     fclose(out);
05013 }
05014 }
05015
05016 /*****
05017
05018 void write_station(
05019     const char *filename,
05020     ctl_t * ctl,
05021     atm_t * atm,
05022     double t) {
05023
05024     static FILE *out;
05025
05026     static double rmax2, t0, t1, x0[3], x1[3];
05027
05028     /* Set timer... */
05029     SELECT_TIMER("WRITE_STATION", NVTX_WRITE);
05030
05031     /* Init... */
05032     if (t == ctl->t_start) {
05033
05034         /* Write info... */
05035         printf("Write station data: %s\n", filename);
05036
05037         /* Create new file... */
05038         if (!(out = fopen(filename, "w")))
05039             ERRMSG("Cannot create file!");
05040
05041         /* Write header... */
05042         fprintf(out,
05043             "# $1 = time [s]\n"
05044             "# $2 = altitude [km]\n"
05045             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
05046         for (int iq = 0; iq < ctl->nq; iq++)
05047             fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
05048                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05049         fprintf(out, "\n");
05050
05051         /* Set geolocation and search radius... */
05052         geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
05053         rmax2 = SQR(ctl->stat_r);
05054     }
05055
05056     /* Set time interval for output... */
05057     t0 = t - 0.5 * ctl->dt_mod;
05058     t1 = t + 0.5 * ctl->dt_mod;
05059
05060     /* Loop over air parcels... */
05061     for (int ip = 0; ip < atm->np; ip++) {

```

```

05062
05063     /* Check time... */
05064     if (atm->time[ip] < t0 || atm->time[ip] > t1)
05065         continue;
05066
05067     /* Check station flag... */
05068     if (ctl->qnt_stat >= 0)
05069         if (atm->q[ctl->qnt_stat][ip])
05070             continue;
05071
05072     /* Get Cartesian coordinates... */
05073     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
05074
05075     /* Check horizontal distance... */
05076     if (DIST2(x0, x1) > rmax2)
05077         continue;
05078
05079     /* Set station flag... */
05080     if (ctl->qnt_stat >= 0)
05081         atm->q[ctl->qnt_stat][ip] = 1;
05082
05083     /* Write data... */
05084     fprintf(out, "%.2f %g %g %g",
05085            atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
05086     for (int iq = 0; iq < ctl->nq; iq++) {
05087         fprintf(out, " ");
05088         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
05089     }
05090     fprintf(out, "\n");
05091 }
05092
05093     /* Close file... */
05094     if (t == ctl->t_stop)
05095         fclose(out);
05096 }

```

## 5.23 libtrac.h File Reference

### Data Structures

- struct [ctl\\_t](#)  
*Control parameters.*
- struct [atm\\_t](#)  
*Atmospheric data.*
- struct [cache\\_t](#)  
*Cache data.*
- struct [met\\_t](#)  
*Meteorological data.*

### Functions

- void [cart2geo](#) (double \*x, double \*z, double \*lon, double \*lat)  
*Convert Cartesian coordinates to geolocation.*
- int [check\\_finite](#) (const double x)  
*Check if x is finite.*
- double [clim\\_hno3](#) (double t, double lat, double p)  
*Climatology of HNO3 volume mixing ratios.*
- double [clim\\_oh](#) (double t, double lat, double p)  
*Climatology of OH number concentrations.*
- double [clim\\_tropo](#) (double t, double lat)  
*Climatology of tropopause pressure.*
- void [day2doy](#) (int year, int mon, int day, int \*doy)  
*Get day of year from date.*
- void [doy2day](#) (int year, int doy, int \*mon, int \*day)

- Get date from day of year.*

  - void `geo2cart` (double z, double lon, double lat, double \*x)
- Convert geolocation to Cartesian coordinates.*

  - void `get_met` (ctl\_t \*ctl, double t, met\_t \*\*met0, met\_t \*\*met1)
- Get meteorological data for given time step.*

  - void `get_met_help` (double t, int direct, char \*metbase, double dt\_met, char \*filename)
- Get meteorological data for time step.*

  - void `get_met_replace` (char \*orig, char \*search, char \*repl)
- Replace template strings in filename.*

  - void `intpol_met_space_3d` (met\_t \*met, float array[EX][EY][EP], double p, double lon, double lat, double \*var, int \*ci, double \*cw, int init)
- Spatial interpolation of meteorological data.*

  - void `intpol_met_space_2d` (met\_t \*met, float array[EX][EY], double lon, double lat, double \*var, int \*ci, double \*cw, int init)
- Spatial interpolation of meteorological data.*

  - void `intpol_met_time_3d` (met\_t \*met0, float array0[EX][EY][EP], met\_t \*met1, float array1[EX][EY][EP], double ts, double p, double lon, double lat, double \*var, int \*ci, double \*cw, int init)
- Temporal interpolation of meteorological data.*

  - void `intpol_met_time_2d` (met\_t \*met0, float array0[EX][EY], met\_t \*met1, float array1[EX][EY], double ts, double lon, double lat, double \*var, int \*ci, double \*cw, int init)
- 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.*

  - double `lapse_rate` (double t, double h2o)
- Calculate moist adiabatic lapse rate.*

  - int `locate_irr` (double \*xx, int n, double x)
- Find array index for irregular grid.*

  - int `locate_reg` (double \*xx, int n, double x)
- Find array index for regular grid.*

  - double `nat_temperature` (double p, double h2o, double hno3)
- Calculate NAT existence temperature.*

  - int `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.*

  - int `read_met` (ctl\_t \*ctl, char \*filename, met\_t \*met)
- Read meteorological data file.*

  - void `read_metCAPE` (met\_t \*met)
- Calculate convective available potential energy.*

  - void `read_met_cloud` (met\_t \*met)
- Calculate cloud properties.*

  - void `read_met_detrend` (ctl\_t \*ctl, met\_t \*met)
- Apply detrending method to temperature and winds.*

  - void `read_met_extrapolate` (met\_t \*met)
- Extrapolate meteorological data at lower boundary.*

  - void `read_met_geopot` (ctl\_t \*ctl, met\_t \*met)
- Calculate geopotential heights.*

  - void `read_met_grid` (char \*filename, int ncid, ctl\_t \*ctl, met\_t \*met)
- Read coordinates of meteorological data.*

  - int `read_met_help_3d` (int ncid, char \*varname, char \*varname2, met\_t \*met, float dest[EX][EY][EP], float scl)

*Read and convert 3D variable from meteorological data file.*

- int [read\\_met\\_help\\_2d](#) (int ncid, char \*varname, char \*varname2, [met\\_t](#) \*met, float dest[EX][EY], float scl)

*Read and convert 2D variable from meteorological data file.*

- void [read\\_met\\_levels](#) (int ncid, [ctl\\_t](#) \*ctl, [met\\_t](#) \*met)
- 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.*

- void [read\\_met\\_pv](#) ([met\\_t](#) \*met)

*Calculate potential vorticity.*

- void [read\\_met\\_sample](#) ([ctl\\_t](#) \*ctl, [met\\_t](#) \*met)

*Downsampling of meteorological data.*

- void [read\\_met\\_surface](#) (int ncid, [met\\_t](#) \*met)

*Read surface data.*

- void [read\\_met\\_tropo](#) ([ctl\\_t](#) \*ctl, [met\\_t](#) \*met)

*Calculate tropopause data.*

- 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.*

- double [sedi](#) (double p, double T, double r\_p, double rho\_p)

*Calculate sedimentation velocity.*

- void [spline](#) (double \*x, double \*y, int n, double \*x2, double \*y2, int n2)

*Spline interpolation.*

- double [stddev](#) (double \*data, int n)

*Calculate standard deviation.*

- 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 output)

*Measure wall-clock time.*

- 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\\_sample](#) (const char \*filename, [ctl\\_t](#) \*ctl, [met\\_t](#) \*met0, [met\\_t](#) \*met1, [atm\\_t](#) \*atm, double t)

*Write sample data.*

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

*Write station data.*

### 5.23.1 Function Documentation

**5.23.1.1 cart2geo()** 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 = NORM(x);
00036     *lat = asin(x[2] / radius) * 180. / M_PI;
00037     *lon = atan2(x[1], x[0]) * 180. / M_PI;
00038     *z = radius - RE;
00039 }

```

**5.23.1.2 check\_finite()** int check\_finite (

```

    const double x )

```

Check if x is finite.

**5.23.1.3 clim\_hno3()** double clim\_hno3 (

```

    double t,
    double lat,
    double p )

```

Climatology of HNO3 volume mixing ratios.

Definition at line 295 of file [libtrac.c](#).

```

00298     {
00299
00300     /* Get seconds since begin of year... */
00301     double sec = FMOD(t, 365.25 * 86400.);
00302     while (sec < 0)
00303         sec += 365.25 * 86400.;
00304
00305     /* Check pressure... */
00306     if (p < clim_hno3_ps[0])
00307         p = clim_hno3_ps[0];
00308     else if (p > clim_hno3_ps[9])
00309         p = clim_hno3_ps[9];
00310
00311     /* Check latitude... */
00312     if (lat < clim_hno3_lats[0])
00313         lat = clim_hno3_lats[0];
00314     else if (lat > clim_hno3_lats[17])
00315         lat = clim_hno3_lats[17];
00316
00317     /* Get indices... */
00318     int isec = locate_irr(clim_hno3_secs, 12, sec);
00319     int ilat = locate_reg(clim_hno3_lats, 18, lat);
00320     int ip = locate_irr(clim_hno3_ps, 10, p);
00321
00322     /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
00323     double aux00 = LIN(clim_hno3_ps[ip],
00324                        clim_hno3_var[isec][ilat][ip],
00325                        clim_hno3_ps[ip + 1],
00326                        clim_hno3_var[isec][ilat][ip + 1], p);
00327     double aux01 = LIN(clim_hno3_ps[ip],
00328                        clim_hno3_var[isec][ilat + 1][ip],
00329                        clim_hno3_ps[ip + 1],
00330                        clim_hno3_var[isec][ilat + 1][ip + 1], p);
00331     double aux10 = LIN(clim_hno3_ps[ip],
00332                        clim_hno3_var[isec + 1][ilat][ip],
00333                        clim_hno3_ps[ip + 1],
00334                        clim_hno3_var[isec + 1][ilat][ip + 1], p);

```

```

00335     double aux11 = LIN(clim_hno3_ps[ip],
00336                       clim_hno3_var[isec + 1][ilat + 1][ip],
00337                       clim_hno3_ps[ip + 1],
00338                       clim_hno3_var[isec + 1][ilat + 1][ip + 1], p);
00339     aux00 = LIN(clim_hno3_lats[ilat], aux00,
00340               clim_hno3_lats[ilat + 1], aux01, lat);
00341     aux11 = LIN(clim_hno3_lats[ilat], aux10,
00342               clim_hno3_lats[ilat + 1], aux11, lat);
00343     aux00 = LIN(clim_hno3_secs[isec], aux00,
00344               clim_hno3_secs[isec + 1], aux11, sec);
00345     return GSL_MAX(aux00, 0.0);
00346 }

```

**5.23.1.4 clim\_oh()** double clim\_oh (  
double t,  
double lat,  
double p )

Climatology of OH number concentrations.

Definition at line 1329 of file libtrac.c.

```

01332     {
01333
01334     /* Get seconds since begin of year... */
01335     double sec = FMOD(t, 365.25 * 86400.);
01336     while (sec < 0)
01337         sec += 365.25 * 86400.;
01338
01339     /* Check pressure... */
01340     if (p < clim_oh_ps[0])
01341         p = clim_oh_ps[0];
01342     else if (p > clim_oh_ps[33])
01343         p = clim_oh_ps[33];
01344
01345     /* Check latitude... */
01346     if (lat < clim_oh_lats[0])
01347         lat = clim_oh_lats[0];
01348     else if (lat > clim_oh_lats[17])
01349         lat = clim_oh_lats[17];
01350
01351     /* Get indices... */
01352     int isec = locate_irr(clim_oh_secs, 12, sec);
01353     int ilat = locate_reg(clim_oh_lats, 18, lat);
01354     int ip = locate_irr(clim_oh_ps, 34, p);
01355
01356     /* Interpolate OH climatology (Pommrich et al., 2014)... */
01357     double aux00 = LIN(clim_oh_ps[ip],
01358                     clim_oh_var[isec][ilat][ip],
01359                     clim_oh_ps[ip + 1],
01360                     clim_oh_var[isec][ilat][ip + 1], p);
01361     double aux01 = LIN(clim_oh_ps[ip],
01362                     clim_oh_var[isec][ilat + 1][ip],
01363                     clim_oh_ps[ip + 1],
01364                     clim_oh_var[isec][ilat + 1][ip + 1], p);
01365     double aux10 = LIN(clim_oh_ps[ip],
01366                     clim_oh_var[isec + 1][ilat][ip],
01367                     clim_oh_ps[ip + 1],
01368                     clim_oh_var[isec + 1][ilat][ip + 1], p);
01369     double aux11 = LIN(clim_oh_ps[ip],
01370                     clim_oh_var[isec + 1][ilat + 1][ip],
01371                     clim_oh_ps[ip + 1],
01372                     clim_oh_var[isec + 1][ilat + 1][ip + 1], p);
01373     aux00 = LIN(clim_oh_lats[ilat], aux00, clim_oh_lats[ilat + 1], aux01, lat);
01374     aux11 = LIN(clim_oh_lats[ilat], aux10, clim_oh_lats[ilat + 1], aux11, lat);
01375     aux00 = LIN(clim_oh_secs[isec], aux00, clim_oh_secs[isec + 1], aux11, sec);
01376     return GSL_MAX(1e6 * aux00, 0.0);
01377 }

```



```

5.23.1.5 clim_tropo() double clim_tropo (
    double t,
    double lat )

```

Climatology of tropopause pressure.

Definition at line 1510 of file [libtrac.c](#).

```

01512     {
01513
01514     /* Get seconds since begin of year... */
01515     double sec = FMOD(t, 365.25 * 86400.);
01516     while (sec < 0)
01517         sec += 365.25 * 86400.;
01518
01519     /* Get indices... */
01520     int isec = locate_irr(clim_tropo_secs, 12, sec);
01521     int ilat = locate_reg(clim_tropo_lats, 73, lat);
01522
01523     /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
01524     double p0 = LIN(clim_tropo_lats[ilat],
01525                     clim_tropo_tps[isec][ilat],
01526                     clim_tropo_lats[ilat + 1],
01527                     clim_tropo_tps[isec][ilat + 1], lat);
01528     double p1 = LIN(clim_tropo_lats[ilat],
01529                     clim_tropo_tps[isec + 1][ilat],
01530                     clim_tropo_lats[ilat + 1],
01531                     clim_tropo_tps[isec + 1][ilat + 1], lat);
01532     return LIN(clim_tropo_secs[isec], p0, clim_tropo_secs[isec + 1], p1, sec);
01533 }

```

Here is the call graph for this function:



```

5.23.1.6 day2doy() void day2doy (
    int year,
    int mon,
    int day,
    int * doy )

```

Get day of year from date.

Definition at line 1537 of file [libtrac.c](#).

```

01541     {
01542
01543     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01544     int d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01545
01546     /* Get day of year... */
01547     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
01548         *doy = d01[mon - 1] + day - 1;
01549     else
01550         *doy = d0[mon - 1] + day - 1;
01551 }

```

**5.23.1.7 doy2day()** void doy2day (  
     int year,  
     int doy,  
     int \* mon,  
     int \* day )

Get date from day of year.

Definition at line 1555 of file libtrac.c.

```
01559     {
01560
01561     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01562     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01563     int i;
01564
01565     /* Get month and day... */
01566     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
01567         for (i = 11; i >= 0; i--)
01568             if (d0l[i] <= doy)
01569                 break;
01570         *mon = i + 1;
01571         *day = doy - d0l[i] + 1;
01572     } else {
01573         for (i = 11; i >= 0; i--)
01574             if (d0[i] <= doy)
01575                 break;
01576         *mon = i + 1;
01577         *day = doy - d0[i] + 1;
01578     }
01579 }
```

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

Convert geolocation to Cartesian coordinates.

Definition at line 1583 of file libtrac.c.

```
01587     {
01588
01589     double radius = z + RE;
01590     x[0] = radius * cos(lat / 180. * M_PI) * cos(lon / 180. * M_PI);
01591     x[1] = radius * cos(lat / 180. * M_PI) * sin(lon / 180. * M_PI);
01592     x[2] = radius * sin(lat / 180. * M_PI);
01593 }
```

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

Get meteorological data for given time step.

Definition at line 1597 of file libtrac.c.

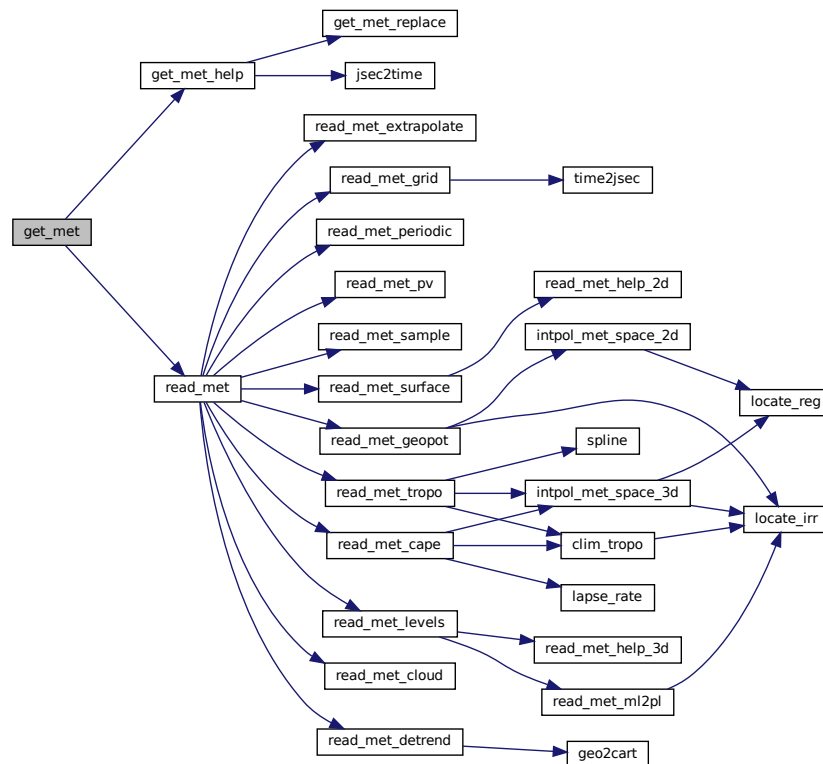
```
01601     {
01602
01603     static int init, ip, ix, iy;
01604
01605     met_t *mets;
01606
01607     char filename[LEN];
```

```

01608
01609 /* Set timer... */
01610 SELECT_TIMER("GET_MET", NVTX_READ);
01611
01612 /* Init... */
01613 if (t == ctl->t_start || !init) {
01614     init = 1;
01615
01616     get_met_help(t, -1, ctl->metbase, ctl->dt_met, filename);
01617     if (!read_met(ctl, filename, *met0))
01618         ERRMSG("Cannot open file!");
01619
01620     get_met_help(t + 1.0 * ctl->direction, 1, ctl->metbase, ctl->dt_met,
01621                 filename);
01622     if (!read_met(ctl, filename, *met1))
01623         ERRMSG("Cannot open file!");
01624 #ifdef _OPENACC
01625     met_t *met0up = *met0;
01626     met_t *met1up = *met1;
01627 #pragma acc update device(met0up[:1],met1up[:1])
01628 #endif
01629 }
01630
01631 /* Read new data for forward trajectories... */
01632 if (t > (*met1)->time && ctl->direction == 1) {
01633     mets = *met1;
01634     *met1 = *met0;
01635     *met0 = mets;
01636     get_met_help(t, 1, ctl->metbase, ctl->dt_met, filename);
01637     if (!read_met(ctl, filename, *met1))
01638         ERRMSG("Cannot open file!");
01639 #ifdef _OPENACC
01640     met_t *met1up = *met1;
01641 #pragma acc update device(met1up[:1])
01642 #endif
01643 }
01644
01645 /* Read new data for backward trajectories... */
01646 if (t < (*met0)->time && ctl->direction == -1) {
01647     mets = *met1;
01648     *met1 = *met0;
01649     *met0 = mets;
01650     get_met_help(t, -1, ctl->metbase, ctl->dt_met, filename);
01651     if (!read_met(ctl, filename, *met0))
01652         ERRMSG("Cannot open file!");
01653 #ifdef _OPENACC
01654     met_t *met0up = *met0;
01655 #pragma acc update device(met0up[:1])
01656 #endif
01657 }
01658
01659 /* Check that grids are consistent... */
01660 if ((*met0)->nx != (*met1)->nx
01661     || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
01662     ERRMSG("Meteo grid dimensions do not match!");
01663 for (ix = 0; ix < (*met0)->nx; ix++)
01664     if (fabs((*met0)->lon[ix] - (*met1)->lon[ix]) > 0.001)
01665         ERRMSG("Meteo grid longitudes do not match!");
01666 for (iy = 0; iy < (*met0)->ny; iy++)
01667     if (fabs((*met0)->lat[iy] - (*met1)->lat[iy]) > 0.001)
01668         ERRMSG("Meteo grid latitudes do not match!");
01669 for (ip = 0; ip < (*met0)->np; ip++)
01670     if (fabs((*met0)->p[ip] - (*met1)->p[ip]) > 0.001)
01671         ERRMSG("Meteo grid pressure levels do not match!");
01672 }

```

Here is the call graph for this function:



**5.23.1.10 get\_met\_help()** void get\_met\_help (

```

double t,
int direct,
char * metbase,
double dt_met,
char * filename )

```

Get meteorological data for time step.

Definition at line 1676 of file libtrac.c.

```

01681     {
01682
01683     char repl[LEN];
01684
01685     double t6, r;
01686
01687     int year, mon, day, hour, min, sec;
01688
01689     /* Round time to fixed intervals... */
01690     if (direct == -1)
01691         t6 = floor(t / dt_met) * dt_met;
01692     else
01693         t6 = ceil(t / dt_met) * dt_met;
01694
01695     /* Decode time... */
01696     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
01697
01698     /* Set filename... */
01699     sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
01700     sprintf(repl, "%d", year);

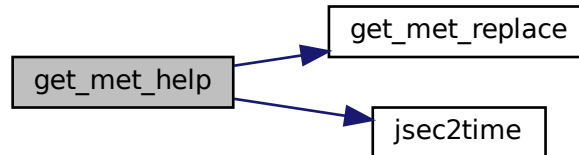
```

```

01701  get_met_replace(filename, "YYYY", repl);
01702  sprintf(repl, "%02d", mon);
01703  get_met_replace(filename, "MM", repl);
01704  sprintf(repl, "%02d", day);
01705  get_met_replace(filename, "DD", repl);
01706  sprintf(repl, "%02d", hour);
01707  get_met_replace(filename, "HH", repl);
01708  }

```

Here is the call graph for this function:



**5.23.1.11 get\_met\_replace()** void get\_met\_replace (  
char \* orig,  
char \* search,  
char \* repl )

Replace template strings in filename.

Definition at line 1712 of file libtrac.c.

```

01715  {
01716
01717  char buffer[LEN], *ch;
01718
01719  /* Iterate... */
01720  for (int i = 0; i < 3; i++) {
01721
01722  /* Replace sub-string... */
01723  if (!(ch = strstr(orig, search)))
01724  return;
01725  strncpy(buffer, orig, (size_t)(ch - orig));
01726  buffer[ch - orig] = 0;
01727  sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
01728  orig[0] = 0;
01729  strcpy(orig, buffer);
01730  }
01731 }

```

**5.23.1.12 intpol\_met\_space\_3d()** void intpol\_met\_space\_3d (  
met\_t \* met,  
float array[EX][EY][EP],  
double p,  
double lon,  
double lat,  
double \* var,  
int \* ci,

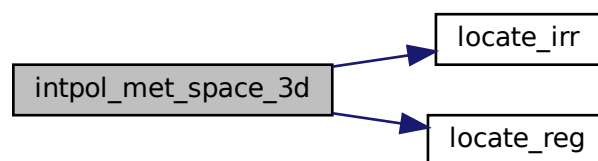
```
double * cw,
int init )
```

Spatial interpolation of meteorological data.

Definition at line 1735 of file libtrac.c.

```
01744     {
01745
01746     /* Check longitude... */
01747     if (met->lon[met->nx - 1] > 180 && lon < 0)
01748         lon += 360;
01749
01750     /* Get interpolation indices and weights... */
01751     if (init) {
01752         ci[0] = locate_irr(met->p, met->np, p);
01753         ci[1] = locate_reg(met->lon, met->nx, lon);
01754         ci[2] = locate_reg(met->lat, met->ny, lat);
01755         cw[0] = (met->p[ci[0] + 1] - p)
01756             / (met->p[ci[0] + 1] - met->p[ci[0]]);
01757         cw[1] = (met->lon[ci[1] + 1] - lon)
01758             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01759         cw[2] = (met->lat[ci[2] + 1] - lat)
01760             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01761     }
01762
01763     /* Interpolate vertically... */
01764     double aux00 =
01765         cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
01766         + array[ci[1]][ci[2]][ci[0] + 1];
01767     double aux01 =
01768         cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
01769             array[ci[1]][ci[2] + 1][ci[0] + 1])
01770         + array[ci[1]][ci[2] + 1][ci[0] + 1];
01771     double aux10 =
01772         cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
01773             array[ci[1] + 1][ci[2]][ci[0] + 1])
01774         + array[ci[1] + 1][ci[2]][ci[0] + 1];
01775     double aux11 =
01776         cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
01777             array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
01778         + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];
01779
01780     /* Interpolate horizontally... */
01781     aux00 = cw[2] * (aux00 - aux01) + aux01;
01782     aux11 = cw[2] * (aux10 - aux11) + aux11;
01783     *var = cw[1] * (aux00 - aux11) + aux11;
01784 }
```

Here is the call graph for this function:



**5.23.1.13 intpol\_met\_space\_2d()** void intpol\_met\_space\_2d (  
     met\_t \* met,  
     float array[EX][EY],  
     double lon,

```

double lat,
double * var,
int * ci,
double * cw,
int init )

```

Spatial interpolation of meteorological data.

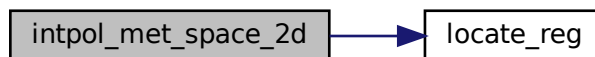
Definition at line 1789 of file libtrac.c.

```

01797     {
01798
01799     /* Check longitude... */
01800     if (met->lon[met->nx - 1] > 180 && lon < 0)
01801         lon += 360;
01802
01803     /* Get interpolation indices and weights... */
01804     if (init) {
01805         ci[1] = locate_reg(met->lon, met->nx, lon);
01806         ci[2] = locate_reg(met->lat, met->ny, lat);
01807         cw[1] = (met->lon[ci[1] + 1] - lon)
01808             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01809         cw[2] = (met->lat[ci[2] + 1] - lat)
01810             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01811     }
01812
01813     /* Set variables... */
01814     double aux00 = array[ci[1]][ci[2]];
01815     double aux01 = array[ci[1]][ci[2] + 1];
01816     double aux10 = array[ci[1] + 1][ci[2]];
01817     double aux11 = array[ci[1] + 1][ci[2] + 1];
01818
01819     /* Interpolate horizontally... */
01820     if (isfinite(aux00) && isfinite(aux01))
01821         aux00 = cw[2] * (aux00 - aux01) + aux01;
01822     else if (cw[2] < 0.5)
01823         aux00 = aux01;
01824     if (isfinite(aux10) && isfinite(aux11))
01825         aux11 = cw[2] * (aux10 - aux11) + aux11;
01826     else if (cw[2] > 0.5)
01827         aux11 = aux10;
01828     if (isfinite(aux00) && isfinite(aux11))
01829         *var = cw[1] * (aux00 - aux11) + aux11;
01830     else {
01831         if (cw[1] > 0.5)
01832             *var = aux00;
01833         else
01834             *var = aux11;
01835     }
01836 }

```

Here is the call graph for this function:



**5.23.1.14 intpol\_met\_time\_3d()** void intpol\_met\_time\_3d (

```

    met_t * met0,
    float array0[EX][EY][EP],
    met_t * met1,
    float array1[EX][EY][EP],

```

```

double ts,
double p,
double lon,
double lat,
double * var,
int * ci,
double * cw,
int init )

```

Temporal interpolation of meteorological data.

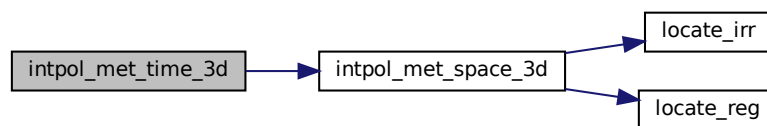
Definition at line 1840 of file libtrac.c.

```

1852     {
1853
1854     double var0, var1, wt;
1855
1856     /* Spatial interpolation... */
1857     intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
1858     intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, init);
1859
1860     /* Get weighting factor... */
1861     wt = (met1->time - ts) / (met1->time - met0->time);
1862
1863     /* Interpolate... */
1864     *var = wt * (var0 - var1) + var1;
1865 }

```

Here is the call graph for this function:



#### 5.23.1.15 intpol\_met\_time\_2d() void intpol\_met\_time\_2d (

```

    met_t * met0,
    float array0[EX][EY],
    met_t * met1,
    float array1[EX][EY],
    double ts,
    double lon,
    double lat,
    double * var,
    int * ci,
    double * cw,
    int init )

```

Temporal interpolation of meteorological data.

Definition at line 1869 of file libtrac.c.

```

1880     {
1881
1882     double var0, var1, wt;
1883
1884     /* Spatial interpolation... */

```

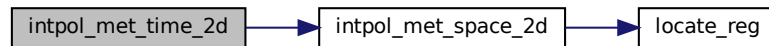


```

01885  intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
01886  intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, init);
01887
01888  /* Get weighting factor... */
01889  wt = (met1->time - ts) / (met1->time - met0->time);
01890
01891  /* Interpolate... */
01892  *var = wt * (var0 - var1) + var1;
01893 }

```

Here is the call graph for this function:



**5.23.1.16 jsec2time()** 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 1897 of file [libtrac.c](#).

```

01905  {
01906
01907  struct tm t0, *t1;
01908
01909  t0.tm_year = 100;
01910  t0.tm_mon = 0;
01911  t0.tm_mday = 1;
01912  t0.tm_hour = 0;
01913  t0.tm_min = 0;
01914  t0.tm_sec = 0;
01915
01916  time_t jsec0 = (time_t) jsec + timegm(&t0);
01917  t1 = gmtime(&jsec0);
01918
01919  *year = t1->tm_year + 1900;
01920  *mon = t1->tm_mon + 1;
01921  *day = t1->tm_mday;
01922  *hour = t1->tm_hour;
01923  *min = t1->tm_min;
01924  *sec = t1->tm_sec;
01925  *remain = jsec - floor(jsec);
01926 }

```

**5.23.1.17 lapse\_rate()** double lapse\_rate (  
double t,  
double h2o )

Calculate moist adiabatic lapse rate.

Definition at line 1930 of file libtrac.c.

```
01932     {
01933
01934     /*
01935      Calculate moist adiabatic lapse rate [K/km] from temperature [K]
01936      and water vapor volume mixing ratio [1].
01937
01938      Reference: https://en.wikipedia.org/wiki/Lapse\_rate
01939     */
01940
01941     const double a = RA * SQR(t), r = SH(h2o) / (1. - SH(h2o));
01942
01943     return 1e3 * G0 * (a + LV * r * t) / (CPD * a + SQR(LV) * r * EPS);
01944 }
```

**5.23.1.18 locate\_irr()** int locate\_irr (  
double \* xx,  
int n,  
double x )

Find array index for irregular grid.

Definition at line 1948 of file libtrac.c.

```
01951     {
01952
01953     int ilo = 0;
01954     int ihi = n - 1;
01955     int i = (ihi + ilo) » 1;
01956
01957     if (xx[i] < xx[i + 1])
01958         while (ihi > ilo + 1) {
01959             i = (ihi + ilo) » 1;
01960             if (xx[i] > x)
01961                 ihi = i;
01962             else
01963                 ilo = i;
01964         } else
01965         while (ihi > ilo + 1) {
01966             i = (ihi + ilo) » 1;
01967             if (xx[i] <= x)
01968                 ihi = i;
01969             else
01970                 ilo = i;
01971         }
01972
01973     return ilo;
01974 }
```

**5.23.1.19 locate\_reg()** int locate\_reg (  
double \* xx,  
int n,  
double x )

Find array index for regular grid.

Definition at line 1978 of file libtrac.c.

```
01981     {
01982
01983     /* Calculate index... */
01984     int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
```

```

01985
01986  /* Check range... */
01987  if (i < 0)
01988      i = 0;
01989  else if (i >= n - 2)
01990      i = n - 2;
01991
01992  return i;
01993 }

```

**5.23.1.20 nat\_temperature()** double nat\_temperature (

double p,

double h2o,

double hno3 )

Calculate NAT existence temperature.

Definition at line 1997 of file libtrac.c.

```

02000  {
02001
02002  double p_hno3 = hno3 * p / 1.333224;
02003  double p_h2o = h2o * p / 1.333224;
02004  double a = 0.009179 - 0.00088 * log10(p_h2o);
02005  double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
02006  double c = -11397.0 / a;
02007  double tnat = (-b + sqrt(b * b - 4. * c)) / 2.;
02008  double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
02009  if (x2 > 0)
02010      tnat = x2;
02011
02012  return tnat;
02013 }

```

**5.23.1.21 read\_atm()** int read\_atm (

const char \* filename,

ctl\_t \* ctl,

atm\_t \* atm )

Read atmospheric data.

Definition at line 2017 of file libtrac.c.

```

02020  {
02021
02022  FILE *in;
02023
02024  char line[LEN], *tok;
02025
02026  double t0;
02027
02028  int dimid, ip, iq, ncid, varid;
02029
02030  size_t nparts;
02031
02032  /* Set timer... */
02033  SELECT_TIMER("READ_ATM", NVTX_READ);
02034
02035  /* Init... */
02036  atm->np = 0;
02037
02038  /* Write info... */
02039  printf("Read atmospheric data: %s\n", filename);
02040
02041  /* Read ASCII data... */
02042  if (ctl->atm_type == 0) {
02043
02044      /* Open file... */
02045      if (!(in = fopen(filename, "r"))) {
02046          WARN("File not found!");
02047          return 0;

```

```

02048     }
02049
02050     /* Read line... */
02051     while (fgets(line, LEN, in)) {
02052
02053         /* Read data... */
02054         TOK(line, tok, "%lg", atm->time[atm->np]);
02055         TOK(NULL, tok, "%lg", atm->p[atm->np]);
02056         TOK(NULL, tok, "%lg", atm->lon[atm->np]);
02057         TOK(NULL, tok, "%lg", atm->lat[atm->np]);
02058         for (iq = 0; iq < ctl->nq; iq++)
02059             TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
02060
02061         /* Convert altitude to pressure... */
02062         atm->p[atm->np] = P(atm->p[atm->np]);
02063
02064         /* Increment data point counter... */
02065         if ((++atm->np) > NP)
02066             ERRMSG("Too many data points!");
02067     }
02068
02069     /* Close file... */
02070     fclose(in);
02071 }
02072
02073 /* Read binary data... */
02074 else if (ctl->atm_type == 1) {
02075
02076     /* Open file... */
02077     if (!(in = fopen(filename, "r")))
02078         return 0;
02079
02080     /* Read data... */
02081     FREAD(&atm->np, int,
02082          1,
02083          in);
02084     FREAD(atm->time, double,
02085          (size_t) atm->np,
02086          in);
02087     FREAD(atm->p, double,
02088          (size_t) atm->np,
02089          in);
02090     FREAD(atm->lon, double,
02091          (size_t) atm->np,
02092          in);
02093     FREAD(atm->lat, double,
02094          (size_t) atm->np,
02095          in);
02096     for (iq = 0; iq < ctl->nq; iq++)
02097         FREAD(atm->q[iq], double,
02098              (size_t) atm->np,
02099              in);
02100
02101     /* Close file... */
02102     fclose(in);
02103 }
02104
02105 /* Read netCDF data... */
02106 else if (ctl->atm_type == 2) {
02107
02108     /* Open file... */
02109     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02110         return 0;
02111
02112     /* Get dimensions... */
02113     NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
02114     NC(nc_inq_dimlen(ncid, dimid, &nparts));
02115     atm->np = (int) nparts;
02116     if (atm->np > NP)
02117         ERRMSG("Too many particles!");
02118
02119     /* Get time... */
02120     NC(nc_inq_varid(ncid, "time", &varid));
02121     NC(nc_get_var_double(ncid, varid, &t0));
02122     for (ip = 0; ip < atm->np; ip++)
02123         atm->time[ip] = t0;
02124
02125     /* Read geolocations... */
02126     NC(nc_inq_varid(ncid, "PRESS", &varid));
02127     NC(nc_get_var_double(ncid, varid, atm->p));
02128     NC(nc_inq_varid(ncid, "LON", &varid));
02129     NC(nc_get_var_double(ncid, varid, atm->lon));
02130     NC(nc_inq_varid(ncid, "LAT", &varid));
02131     NC(nc_get_var_double(ncid, varid, atm->lat));
02132
02133     /* Read variables... */
02134     if (ctl->qnt_p >= 0)

```

```

02135     if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
02136         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
02137     if (ctl->qnt_t >= 0)
02138         if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
02139             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
02140     if (ctl->qnt_u >= 0)
02141         if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
02142             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
02143     if (ctl->qnt_v >= 0)
02144         if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
02145             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
02146     if (ctl->qnt_w >= 0)
02147         if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
02148             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
02149     if (ctl->qnt_h2o >= 0)
02150         if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
02151             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
02152     if (ctl->qnt_o3 >= 0)
02153         if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
02154             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
02155     if (ctl->qnt_theta >= 0)
02156         if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
02157             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
02158     if (ctl->qnt_pv >= 0)
02159         if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
02160             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
02161
02162     /* Check data... */
02163     for (ip = 0; ip < atm->np; ip++)
02164         if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
02165             || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
02166             || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
02167             || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
02168             || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
02169         atm->time[ip] = GSL_NAN;
02170         atm->p[ip] = GSL_NAN;
02171         atm->lon[ip] = GSL_NAN;
02172         atm->lat[ip] = GSL_NAN;
02173         for (iq = 0; iq < ctl->nq; iq++)
02174             atm->q[iq][ip] = GSL_NAN;
02175     } else {
02176         if (ctl->qnt_h2o >= 0)
02177             atm->q[ctl->qnt_h2o][ip] *= 1.608;
02178         if (ctl->qnt_pv >= 0)
02179             atm->q[ctl->qnt_pv][ip] *= 1e6;
02180         if (atm->lon[ip] > 180)
02181             atm->lon[ip] -= 360;
02182     }
02183
02184     /* Close file... */
02185     NC(nc_close(ncid));
02186 }
02187
02188 /* Error... */
02189 else
02190     ERRMSG("Atmospheric data type not supported!");
02191
02192 /* Check number of points... */
02193 if (atm->np < 1)
02194     ERRMSG("Can not read any data!");
02195
02196 /* Return success... */
02197 return 1;
02198 }

```

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

Read control parameters.

Definition at line 2202 of file libtrac.c.

```

02206     {
02207
02208     /* Set timer... */

```

```

02209 SELECT_TIMER("READ_CTL", NVTX_READ);
02210
02211 /* Write info... */
02212 printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
02213        "(executable: %s | compiled: %s, %s)\n\n",
02214        argv[0], __DATE__, __TIME__);
02215
02216 /* Initialize quantity indices... */
02217 ctl->qnt_ens = -1;
02218 ctl->qnt_m = -1;
02219 ctl->qnt_r = -1;
02220 ctl->qnt_rho = -1;
02221 ctl->qnt_ps = -1;
02222 ctl->qnt_ts = -1;
02223 ctl->qnt_zs = -1;
02224 ctl->qnt_us = -1;
02225 ctl->qnt_vs = -1;
02226 ctl->qnt_pt = -1;
02227 ctl->qnt_tt = -1;
02228 ctl->qnt_zt = -1;
02229 ctl->qnt_h2ot = -1;
02230 ctl->qnt_z = -1;
02231 ctl->qnt_p = -1;
02232 ctl->qnt_t = -1;
02233 ctl->qnt_u = -1;
02234 ctl->qnt_v = -1;
02235 ctl->qnt_w = -1;
02236 ctl->qnt_h2o = -1;
02237 ctl->qnt_o3 = -1;
02238 ctl->qnt_lwc = -1;
02239 ctl->qnt_iwc = -1;
02240 ctl->qnt_pc = -1;
02241 ctl->qnt_cl = -1;
02242 ctl->qnt_plcl = -1;
02243 ctl->qnt_plfc = -1;
02244 ctl->qnt_pel = -1;
02245 ctl->qnt_cape = -1;
02246 ctl->qnt_hno3 = -1;
02247 ctl->qnt_oh = -1;
02248 ctl->qnt_psat = -1;
02249 ctl->qnt_psize = -1;
02250 ctl->qnt_pw = -1;
02251 ctl->qnt_sh = -1;
02252 ctl->qnt_rh = -1;
02253 ctl->qnt_rhice = -1;
02254 ctl->qnt_theta = -1;
02255 ctl->qnt_tvirt = -1;
02256 ctl->qnt_lapse = -1;
02257 ctl->qnt_vh = -1;
02258 ctl->qnt_vz = -1;
02259 ctl->qnt_pv = -1;
02260 ctl->qnt_tdew = -1;
02261 ctl->qnt_tice = -1;
02262 ctl->qnt_tsts = -1;
02263 ctl->qnt_tnat = -1;
02264 ctl->qnt_stat = -1;
02265
02266 /* Read quantities... */
02267 ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
02268 if (ctl->nq > NQ)
02269     ERRMSG("Too many quantities!");
02270 for (int iq = 0; iq < ctl->nq; iq++) {
02271
02272     /* Read quantity name and format... */
02273     scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
02274     scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
02275             ctl->qnt_format[iq]);
02276
02277     /* Try to identify quantity... */
02278     if (strcasecmp(ctl->qnt_name[iq], "ens") == 0) {
02279         ctl->qnt_ens = iq;
02280         sprintf(ctl->qnt_unit[iq], "-");
02281     } else if (strcasecmp(ctl->qnt_name[iq], "m") == 0) {
02282         ctl->qnt_m = iq;
02283         sprintf(ctl->qnt_unit[iq], "kg");
02284     } else if (strcasecmp(ctl->qnt_name[iq], "r") == 0) {
02285         ctl->qnt_r = iq;
02286         sprintf(ctl->qnt_unit[iq], "m");
02287     } else if (strcasecmp(ctl->qnt_name[iq], "rho") == 0) {
02288         ctl->qnt_rho = iq;
02289         sprintf(ctl->qnt_unit[iq], "kg/m^3");
02290     } else if (strcasecmp(ctl->qnt_name[iq], "ps") == 0) {
02291         ctl->qnt_ps = iq;
02292         sprintf(ctl->qnt_unit[iq], "hPa");
02293     } else if (strcasecmp(ctl->qnt_name[iq], "pt") == 0) {
02294         ctl->qnt_pt = iq;
02295         sprintf(ctl->qnt_unit[iq], "hPa");
02296     }
02297 }

```

```

02296     } else if (strcasecmp(ctl->qnt_name[iq], "tt") == 0) {
02297         ctl->qnt_tt = iq;
02298         sprintf(ctl->qnt_unit[iq], "K");
02299     } else if (strcasecmp(ctl->qnt_name[iq], "zt") == 0) {
02300         ctl->qnt_zt = iq;
02301         sprintf(ctl->qnt_unit[iq], "km");
02302     } else if (strcasecmp(ctl->qnt_name[iq], "h2ot") == 0) {
02303         ctl->qnt_h2ot = iq;
02304         sprintf(ctl->qnt_unit[iq], "ppv");
02305     } else if (strcasecmp(ctl->qnt_name[iq], "z") == 0) {
02306         ctl->qnt_z = iq;
02307         sprintf(ctl->qnt_unit[iq], "km");
02308     } else if (strcasecmp(ctl->qnt_name[iq], "p") == 0) {
02309         ctl->qnt_p = iq;
02310         sprintf(ctl->qnt_unit[iq], "hPa");
02311     } else if (strcasecmp(ctl->qnt_name[iq], "t") == 0) {
02312         ctl->qnt_t = iq;
02313         sprintf(ctl->qnt_unit[iq], "K");
02314     } else if (strcasecmp(ctl->qnt_name[iq], "u") == 0) {
02315         ctl->qnt_u = iq;
02316         sprintf(ctl->qnt_unit[iq], "m/s");
02317     } else if (strcasecmp(ctl->qnt_name[iq], "v") == 0) {
02318         ctl->qnt_v = iq;
02319         sprintf(ctl->qnt_unit[iq], "m/s");
02320     } else if (strcasecmp(ctl->qnt_name[iq], "w") == 0) {
02321         ctl->qnt_w = iq;
02322         sprintf(ctl->qnt_unit[iq], "hPa/s");
02323     } else if (strcasecmp(ctl->qnt_name[iq], "h2o") == 0) {
02324         ctl->qnt_h2o = iq;
02325         sprintf(ctl->qnt_unit[iq], "ppv");
02326     } else if (strcasecmp(ctl->qnt_name[iq], "o3") == 0) {
02327         ctl->qnt_o3 = iq;
02328         sprintf(ctl->qnt_unit[iq], "ppv");
02329     } else if (strcasecmp(ctl->qnt_name[iq], "lwc") == 0) {
02330         ctl->qnt_lwc = iq;
02331         sprintf(ctl->qnt_unit[iq], "kg/kg");
02332     } else if (strcasecmp(ctl->qnt_name[iq], "iwc") == 0) {
02333         ctl->qnt_iwc = iq;
02334         sprintf(ctl->qnt_unit[iq], "kg/kg");
02335     } else if (strcasecmp(ctl->qnt_name[iq], "pc") == 0) {
02336         ctl->qnt_pc = iq;
02337         sprintf(ctl->qnt_unit[iq], "hPa");
02338     } else if (strcasecmp(ctl->qnt_name[iq], "cl") == 0) {
02339         ctl->qnt_cl = iq;
02340         sprintf(ctl->qnt_unit[iq], "kg/m^2");
02341     } else if (strcasecmp(ctl->qnt_name[iq], "plcl") == 0) {
02342         ctl->qnt_plcl = iq;
02343         sprintf(ctl->qnt_unit[iq], "hPa");
02344     } else if (strcasecmp(ctl->qnt_name[iq], "plfc") == 0) {
02345         ctl->qnt_plfc = iq;
02346         sprintf(ctl->qnt_unit[iq], "hPa");
02347     } else if (strcasecmp(ctl->qnt_name[iq], "pel") == 0) {
02348         ctl->qnt_pel = iq;
02349         sprintf(ctl->qnt_unit[iq], "hPa");
02350     } else if (strcasecmp(ctl->qnt_name[iq], "cape") == 0) {
02351         ctl->qnt_cape = iq;
02352         sprintf(ctl->qnt_unit[iq], "J/kg");
02353     } else if (strcasecmp(ctl->qnt_name[iq], "hno3") == 0) {
02354         ctl->qnt_hno3 = iq;
02355         sprintf(ctl->qnt_unit[iq], "ppv");
02356     } else if (strcasecmp(ctl->qnt_name[iq], "oh") == 0) {
02357         ctl->qnt_oh = iq;
02358         sprintf(ctl->qnt_unit[iq], "molec/cm^3");
02359     } else if (strcasecmp(ctl->qnt_name[iq], "psat") == 0) {
02360         ctl->qnt_psat = iq;
02361         sprintf(ctl->qnt_unit[iq], "hPa");
02362     } else if (strcasecmp(ctl->qnt_name[iq], "psice") == 0) {
02363         ctl->qnt_psice = iq;
02364         sprintf(ctl->qnt_unit[iq], "hPa");
02365     } else if (strcasecmp(ctl->qnt_name[iq], "pw") == 0) {
02366         ctl->qnt_pw = iq;
02367         sprintf(ctl->qnt_unit[iq], "hPa");
02368     } else if (strcasecmp(ctl->qnt_name[iq], "sh") == 0) {
02369         ctl->qnt_sh = iq;
02370         sprintf(ctl->qnt_unit[iq], "kg/kg");
02371     } else if (strcasecmp(ctl->qnt_name[iq], "rh") == 0) {
02372         ctl->qnt_rh = iq;
02373         sprintf(ctl->qnt_unit[iq], "%");
02374     } else if (strcasecmp(ctl->qnt_name[iq], "rhice") == 0) {
02375         ctl->qnt_rhice = iq;
02376         sprintf(ctl->qnt_unit[iq], "%");
02377     } else if (strcasecmp(ctl->qnt_name[iq], "theta") == 0) {
02378         ctl->qnt_theta = iq;
02379         sprintf(ctl->qnt_unit[iq], "K");
02380     } else if (strcasecmp(ctl->qnt_name[iq], "tvirt") == 0) {
02381         ctl->qnt_tvirt = iq;
02382         sprintf(ctl->qnt_unit[iq], "K");

```

```

02383     } else if (strcasecmp(ctl->qnt_name[iq], "lapse") == 0) {
02384         ctl->qnt_lapse = iq;
02385         sprintf(ctl->qnt_unit[iq], "K/km");
02386     } else if (strcasecmp(ctl->qnt_name[iq], "vh") == 0) {
02387         ctl->qnt_vh = iq;
02388         sprintf(ctl->qnt_unit[iq], "m/s");
02389     } else if (strcasecmp(ctl->qnt_name[iq], "vz") == 0) {
02390         ctl->qnt_vz = iq;
02391         sprintf(ctl->qnt_unit[iq], "m/s");
02392     } else if (strcasecmp(ctl->qnt_name[iq], "pv") == 0) {
02393         ctl->qnt_pv = iq;
02394         sprintf(ctl->qnt_unit[iq], "PVU");
02395     } else if (strcasecmp(ctl->qnt_name[iq], "tdew") == 0) {
02396         ctl->qnt_tdew = iq;
02397         sprintf(ctl->qnt_unit[iq], "K");
02398     } else if (strcasecmp(ctl->qnt_name[iq], "tice") == 0) {
02399         ctl->qnt_tice = iq;
02400         sprintf(ctl->qnt_unit[iq], "K");
02401     } else if (strcasecmp(ctl->qnt_name[iq], "tsts") == 0) {
02402         ctl->qnt_tsts = iq;
02403         sprintf(ctl->qnt_unit[iq], "K");
02404     } else if (strcasecmp(ctl->qnt_name[iq], "tnat") == 0) {
02405         ctl->qnt_tnat = iq;
02406         sprintf(ctl->qnt_unit[iq], "K");
02407     } else if (strcasecmp(ctl->qnt_name[iq], "stat") == 0) {
02408         ctl->qnt_stat = iq;
02409         sprintf(ctl->qnt_unit[iq], "-");
02410     } else
02411         scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
02412 }
02413
02414 /* Time steps of simulation... */
02415 ctl->direction =
02416     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
02417 if (ctl->direction != -1 && ctl->direction != 1)
02418     ERRMSG("Set DIRECTION to -1 or 1!");
02419 ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
02420 ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "180", NULL);
02421
02422 /* Meteorological data... */
02423 scan_ctl(filename, argc, argv, "METBASE", -1, "-", ctl->metbase);
02424 ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
02425 ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
02426 ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
02427 ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
02428 if (ctl->met_dx < 1 || ctl->met_dy < 1 || ctl->met_dp < 1)
02429     ERRMSG("MET_DX, MET_DY, and MET_DP need to be greater than zero!");
02430 ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
02431 ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
02432 ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
02433 if (ctl->met_sx < 1 || ctl->met_sy < 1 || ctl->met_sp < 1)
02434     ERRMSG("MET_SX, MET_SY, and MET_SP need to be greater than zero!");
02435 ctl->met_detrend =
02436     scan_ctl(filename, argc, argv, "MET_DETREND", -1, "-999", NULL);
02437 ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
02438 if (ctl->met_np > EP)
02439     ERRMSG("Too many levels!");
02440 for (int ip = 0; ip < ctl->met_np; ip++)
02441     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
02442 ctl->met_geopot_sx =
02443     (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SX", -1, "6", NULL);
02444 ctl->met_geopot_sy =
02445     (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SY", -1, "4", NULL);
02446 if (ctl->met_geopot_sx < 1 || ctl->met_geopot_sy < 1)
02447     ERRMSG("MET_GEOPOT_SX and MET_GEOPOT_SY need to be greater than zero!");
02448 ctl->met_tropo =
02449     (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
02450 ctl->met_dt_out =
02451     scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
02452
02453 /* Isosurface parameters... */
02454 ctl->isosurf =
02455     (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
02456 scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
02457
02458 /* Diffusion parameters... */
02459 ctl->turb_dx_trop =
02460     scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
02461 ctl->turb_dx_strat =
02462     scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
02463 ctl->turb_dz_trop =
02464     scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
02465 ctl->turb_dz_strat =
02466     scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
02467 ctl->turb_mesox =
02468     scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
02469 ctl->turb_mesoz =

```



```

02470     scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
02471
02472     /* Convection... */
02473     ctl->conv_cape =
02474         scan_ctl(filename, argc, argv, "CONV_CAPE", -1, "-999", NULL);
02475
02476     /* Species parameters... */
02477     scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
02478     if (strcasecmp(ctl->species, "SO2") == 0) {
02479         ctl->molmass = 64.066;
02480         ctl->oh_chem[0] = 3.3e-31; /* (JPL Publication 15-10) */
02481         ctl->oh_chem[1] = 4.3; /* (JPL Publication 15-10) */
02482         ctl->oh_chem[2] = 1.6e-12; /* (JPL Publication 15-10) */
02483         ctl->oh_chem[3] = 0.0; /* (JPL Publication 15-10) */
02484         ctl->wet_depo[2] = 1.3e-2; /* (Sander, 2015) */
02485         ctl->wet_depo[3] = 2900.0; /* (Sander, 2015) */
02486         ctl->wet_depo[6] = 1.3e-2; /* (Sander, 2015) */
02487         ctl->wet_depo[7] = 2900.0; /* (Sander, 2015) */
02488     } else {
02489         ctl->molmass =
02490             scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
02491         ctl->tdec_trop =
02492             scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
02493         ctl->tdec_strat =
02494             scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
02495         for (int ip = 0; ip < 4; ip++)
02496             ctl->oh_chem[ip] =
02497                 scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
02498         for (int ip = 0; ip < 1; ip++)
02499             ctl->dry_depo[ip] =
02500                 scan_ctl(filename, argc, argv, "DRY_DEPO", ip, "0", NULL);
02501         for (int ip = 0; ip < 8; ip++)
02502             ctl->wet_depo[ip] =
02503                 scan_ctl(filename, argc, argv, "WET_DEPO", ip, "0", NULL);
02504     }
02505
02506     /* PSC analysis... */
02507     ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
02508     ctl->psc_hno3 =
02509         scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
02510
02511     /* Output of atmospheric data... */
02512     scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->atm_basename);
02513     scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
02514     ctl->atm_dt_out =
02515         scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
02516     ctl->atm_filter =
02517         (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
02518     ctl->atm_stride =
02519         (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
02520     ctl->atm_type =
02521         (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
02522
02523     /* Output of CSI data... */
02524     scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->csi_basename);
02525     ctl->csi_dt_out =
02526         scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
02527     scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->csi_obsfile);
02528     ctl->csi_obsmin =
02529         scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
02530     ctl->csi_modmin =
02531         scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
02532     ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
02533     ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
02534     ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
02535     ctl->csi_lon0 =
02536         scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
02537     ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
02538     ctl->csi_nx =
02539         (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
02540     ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
02541     ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
02542     ctl->csi_ny =
02543         (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
02544
02545     /* Output of ensemble data... */
02546     scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->ens_basename);
02547
02548     /* Output of grid data... */
02549     scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
02550         ctl->grid_basename);
02551     scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->grid_gpfile);
02552     ctl->grid_dt_out =
02553         scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
02554     ctl->grid_sparse =
02555         (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
02556     ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);

```

```

02557 ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
02558 ctl->grid_nz =
02559     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
02560 ctl->grid_lon0 =
02561     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
02562 ctl->grid_lon1 =
02563     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
02564 ctl->grid_nx =
02565     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
02566 ctl->grid_lat0 =
02567     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
02568 ctl->grid_lat1 =
02569     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
02570 ctl->grid_ny =
02571     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
02572
02573 /* Output of profile data... */
02574 scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
02575         ctl->prof_basename);
02576 scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->prof_obsfile);
02577 ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
02578 ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
02579 ctl->prof_nz =
02580     (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
02581 ctl->prof_lon0 =
02582     scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
02583 ctl->prof_lon1 =
02584     scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
02585 ctl->prof_nx =
02586     (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
02587 ctl->prof_lat0 =
02588     scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
02589 ctl->prof_lat1 =
02590     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
02591 ctl->prof_ny =
02592     (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
02593
02594 /* Output of sample data... */
02595 scan_ctl(filename, argc, argv, "SAMPLE_BASENAME", -1, "-",
02596         ctl->sample_basename);
02597 scan_ctl(filename, argc, argv, "SAMPLE_OBSFILE", -1, "-",
02598         ctl->sample_obsfile);
02599 ctl->sample_dx =
02600     scan_ctl(filename, argc, argv, "SAMPLE_DX", -1, "50", NULL);
02601 ctl->sample_dz =
02602     scan_ctl(filename, argc, argv, "SAMPLE_DZ", -1, "-999", NULL);
02603
02604 /* Output of station data... */
02605 scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
02606         ctl->stat_basename);
02607 ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
02608 ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
02609 ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
02610 }

```

Here is the call graph for this function:



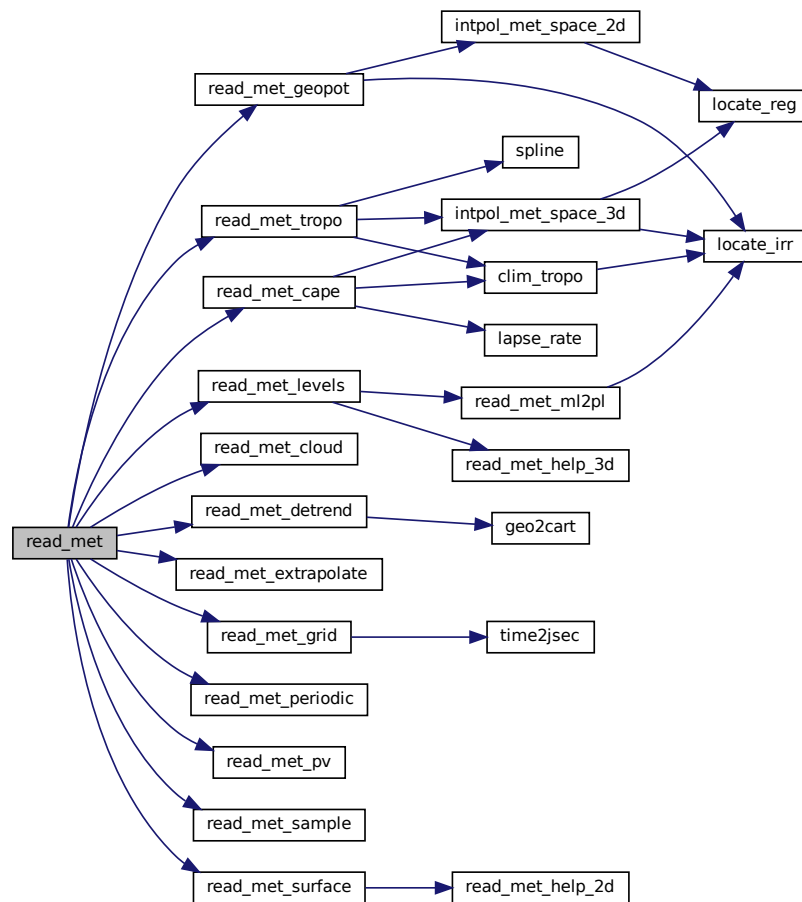
**5.23.1.23 read\_met()** int read\_met (  
     ctl\_t \* ctl,  
     char \* filename,  
     met\_t \* met )

Read meteorological data file.

Definition at line 2614 of file [libtrac.c](#).

```
02617         {
02618
02619     int ncid;
02620
02621     /* Write info... */
02622     printf("Read meteorological data: %s\n", filename);
02623
02624     /* Open netCDF file... */
02625     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
02626         WARN("File not found!");
02627         return 0;
02628     }
02629
02630     /* Read coordinates of meteorological data... */
02631     read_met_grid(filename, ncid, ctl, met);
02632
02633     /* Read meteo data on vertical levels... */
02634     read_met_levels(ncid, ctl, met);
02635
02636     /* Extrapolate data for lower boundary... */
02637     read_met_extrapolate(met);
02638
02639     /* Read surface data... */
02640     read_met_surface(ncid, met);
02641
02642     /* Create periodic boundary conditions... */
02643     read_met_periodic(met);
02644
02645     /* Downsampling... */
02646     read_met_sample(ctl, met);
02647
02648     /* Calculate geopotential heights... */
02649     read_met_geopot(ctl, met);
02650
02651     /* Calculate potential vorticity... */
02652     read_met_pv(met);
02653
02654     /* Calculate tropopause data... */
02655     read_met_tropo(ctl, met);
02656
02657     /* Calculate cloud properties... */
02658     read_met_cloud(met);
02659
02660     /* Calculate convective available potential energy... */
02661     read_met_cape(met);
02662
02663     /* Detrending... */
02664     read_met_detrend(ctl, met);
02665
02666     /* Close file... */
02667     NC(nc_close(ncid));
02668
02669     /* Return success... */
02670     return 1;
02671 }
```

Here is the call graph for this function:



#### 5.23.1.24 read\_met\_cape() `void read_met_cape (met_t * met )`

Calculate convective available potential energy.

Definition at line 2675 of file [libtrac.c](#).

```

02676     {
02677
02678     /* Set timer... */
02679     SELECT_TIMER("READ_MET_CAPE", NVTX_READ);
02680
02681     /* Vertical spacing (about 100 m)... */
02682     const double pfac = 1.01439, dz0 = RI / MA / G0 * log(pfac);
02683
02684     /* Loop over columns... */
02685     #pragma omp parallel for default(shared)
02686     for (int ix = 0; ix < met->nx; ix++)
02687         for (int iy = 0; iy < met->ny; iy++) {
02688
02689         /* Get potential temperature and water vapor vmr at lowest 50 hPa... */
02690         int n = 0;
02691         double h2o = 0, t, theta = 0;
02692         double pbot = GSL_MIN(met->ps[ix][iy], met->p[0]);
02693         double ptop = pbot - 50.;
02694     }
02695 }

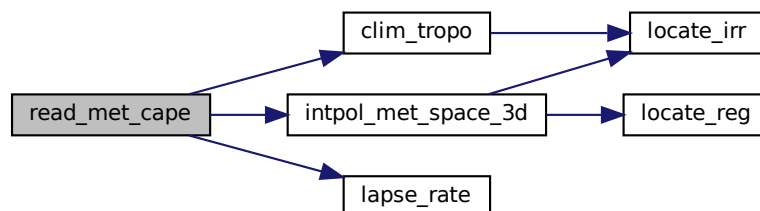
```

```

02694     for (int ip = 0; ip < met->np; ip++) {
02695         if (met->p[ip] <= pbot) {
02696             theta += THETA(met->p[ip], met->t[ix][iy][ip]);
02697             h2o += met->h2o[ix][iy][ip];
02698             n++;
02699         }
02700         if (met->p[ip] < ptop && n > 0)
02701             break;
02702     }
02703     theta /= n;
02704     h2o /= n;
02705
02706     /* Cannot compute anything if water vapor is missing... */
02707     met->plcl[ix][iy] = GSL_NAN;
02708     met->plfc[ix][iy] = GSL_NAN;
02709     met->pel[ix][iy] = GSL_NAN;
02710     met->cape[ix][iy] = GSL_NAN;
02711     if (h2o <= 0)
02712         continue;
02713
02714     /* Find lifted condensation level (LCL)... */
02715     ptop = P(20.);
02716     pbot = met->ps[ix][iy];
02717     do {
02718         met->plcl[ix][iy] = (float) (0.5 * (pbot + ptop));
02719         t = theta / pow(1000. / met->plcl[ix][iy], 0.286);
02720         if (RH(met->plcl[ix][iy], t, h2o) > 100.)
02721             ptop = met->plcl[ix][iy];
02722         else
02723             pbot = met->plcl[ix][iy];
02724     } while (pbot - ptop > 0.1);
02725
02726     /* Calculate level of free convection (LFC), equilibrium level (EL),
02727        and convective available potential energy (CAPE)... */
02728     double dcape = 0, dcape_old, dz, h2o_env, p = met->plcl[ix][iy],
02729         psat, t_env;
02730     ptop = 0.75 * clim_tropo(met->time, met->lat[iy]);
02731     met->cape[ix][iy] = 0;
02732     do {
02733         dz = dz0 * TVIRT(t, h2o);
02734         p /= pfac;
02735         t -= lapse_rate(t, h2o) * dz;
02736         psat = PSAT(t);
02737         h2o = psat / (p - (1. - EPS) * psat);
02738         INTPOL_INIT;
02739         intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
02740                             &t_env, ci, cw, 1);
02741         intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
02742                             &h2o_env, ci, cw, 0);
02743         dcape_old = dcape;
02744         dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
02745             TVIRT(t_env, h2o_env) * dz;
02746         if (dcape > 0) {
02747             met->cape[ix][iy] += (float) dcape;
02748             if (!isfinite(met->plfc[ix][iy]))
02749                 met->plfc[ix][iy] = (float) p;
02750             } else if (dcape_old > 0)
02751                 met->pel[ix][iy] = (float) p;
02752         } while (p > ptop);
02753     }
02754 }

```

Here is the call graph for this function:



**5.23.1.25 read\_met\_cloud()** void read\_met\_cloud (  
     met\_t \* met )

Calculate cloud properties.

Definition at line 2758 of file libtrac.c.

```

02759     {
02760
02761     /* Set timer... */
02762     SELECT_TIMER("READ_MET_CLOUD", NVTX_READ);
02763
02764     /* Loop over columns... */
02765     #pragma omp parallel for default(shared)
02766     for (int ix = 0; ix < met->nx; ix++)
02767         for (int iy = 0; iy < met->ny; iy++) {
02768
02769         /* Init... */
02770         met->pc[ix][iy] = GSL_NAN;
02771         met->cl[ix][iy] = 0;
02772
02773         /* Loop over pressure levels... */
02774         for (int ip = 0; ip < met->np - 1; ip++) {
02775
02776         /* Check pressure... */
02777         if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
02778             continue;
02779
02780         /* Get cloud top pressure ... */
02781         if (met->iwc[ix][iy][ip] > 0 || met->lwc[ix][iy][ip] > 0)
02782             met->pc[ix][iy] = (float) met->p[ip + 1];
02783
02784         /* Get cloud water... */
02785         met->cl[ix][iy] += (float)
02786             (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1]
02787                 + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
02788              * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
02789         }
02790     }
02791 }
```

**5.23.1.26 read\_met\_detrend()** void read\_met\_detrend (  
     ctl\_t \* ctl,  
     met\_t \* met )

Apply detrending method to temperature and winds.

Definition at line 2795 of file libtrac.c.

```

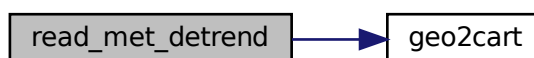
02797     {
02798
02799     met_t *help;
02800
02801     /* Check parameters... */
02802     if (ctl->met_detrend <= 0)
02803         return;
02804
02805     /* Set timer... */
02806     SELECT_TIMER("READ_MET_DETREND", NVTX_READ);
02807
02808     /* Allocate... */
02809     ALLOC(help, met_t, 1);
02810
02811     /* Calculate standard deviation... */
02812     double sigma = ctl->met_detrend / 2.355;
02813     double tssq = 2. * SQR(sigma);
02814
02815     /* Calculate box size in latitude... */
02816     int sy = (int) (3. * DY2DEG(sigma) / fabs(met->lat[1] - met->lat[0]));
02817     sy = GSL_MIN(GSL_MAX(1, sy), met->ny / 2);
02818
02819     /* Calculate background... */
02820     #pragma omp parallel for default(shared)
02821     for (int ix = 0; ix < met->nx; ix++) {
02822         for (int iy = 0; iy < met->ny; iy++) {
02823
02824         /* Calculate Cartesian coordinates... */
02825         double x0[3];
```

```

02826     geo2cart(0.0, met->lon[ix], met->lat[iy], x0);
02827
02828     /* Calculate box size in longitude... */
02829     int sx =
02830         (int) (3. * DX2DEG(sigma, met->lat[iy]) /
02831             fabs(met->lon[1] - met->lon[0]));
02832     sx = GSL_MIN(GSL_MAX(1, sx), met->nx / 2);
02833
02834     /* Init... */
02835     float wsum = 0;
02836     for (int ip = 0; ip < met->np; ip++) {
02837         help->t[ix][iy][ip] = 0;
02838         help->u[ix][iy][ip] = 0;
02839         help->v[ix][iy][ip] = 0;
02840         help->w[ix][iy][ip] = 0;
02841     }
02842
02843     /* Loop over neighboring grid points... */
02844     for (int ix2 = ix - sx; ix2 <= ix + sx; ix2++) {
02845         int ix3 = ix2;
02846         if (ix3 < 0)
02847             ix3 += met->nx;
02848         else if (ix3 >= met->nx)
02849             ix3 -= met->nx;
02850         for (int iy2 = GSL_MAX(iy - sy, 0);
02851             iy2 <= GSL_MIN(iy + sy, met->ny - 1); iy2++) {
02852
02853             /* Calculate Cartesian coordinates... */
02854             double xl[3];
02855             geo2cart(0.0, met->lon[ix3], met->lat[iy2], xl);
02856
02857             /* Calculate weighting factor... */
02858             float w = (float) exp(-DIST2(x0, xl) / tssq);
02859
02860             /* Add data... */
02861             wsum += w;
02862             for (int ip = 0; ip < met->np; ip++) {
02863                 help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip];
02864                 help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip];
02865                 help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip];
02866                 help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip];
02867             }
02868         }
02869     }
02870
02871     /* Normalize... */
02872     for (int ip = 0; ip < met->np; ip++) {
02873         help->t[ix][iy][ip] /= wsum;
02874         help->u[ix][iy][ip] /= wsum;
02875         help->v[ix][iy][ip] /= wsum;
02876         help->w[ix][iy][ip] /= wsum;
02877     }
02878 }
02879 }
02880
02881 /* Subtract background... */
02882 #pragma omp parallel for default(shared)
02883 for (int ix = 0; ix < met->nx; ix++)
02884     for (int iy = 0; iy < met->ny; iy++)
02885         for (int ip = 0; ip < met->np; ip++) {
02886             met->t[ix][iy][ip] -= help->t[ix][iy][ip];
02887             met->u[ix][iy][ip] -= help->u[ix][iy][ip];
02888             met->v[ix][iy][ip] -= help->v[ix][iy][ip];
02889             met->w[ix][iy][ip] -= help->w[ix][iy][ip];
02890         }
02891
02892 /* Free... */
02893 free(help);
02894 }

```

Here is the call graph for this function:



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

Extrapolate meteorological data at lower boundary.

Definition at line 2898 of file libtrac.c.

```

02899     {
02900
02901     int ip, ip0, ix, iy;
02902
02903     /* Set timer... */
02904     SELECT_TIMER("READ_MET_EXTRAPOLATE", NVTX_READ);
02905
02906     /* Loop over columns... */
02907     #pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
02908     for (ix = 0; ix < met->nx; ix++)
02909         for (iy = 0; iy < met->ny; iy++) {
02910
02911         /* Find lowest valid data point... */
02912         for (ip0 = met->np - 1; ip0 >= 0; ip0--)
02913             if (!isfinite(met->t[ix][iy][ip0])
02914                 || !isfinite(met->u[ix][iy][ip0])
02915                 || !isfinite(met->v[ix][iy][ip0])
02916                 || !isfinite(met->w[ix][iy][ip0]))
02917                 break;
02918
02919         /* Extrapolate... */
02920         for (ip = ip0; ip >= 0; ip--) {
02921             met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
02922             met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
02923             met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
02924             met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
02925             met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
02926             met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
02927             met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
02928             met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
02929         }
02930     }
02931 }
```

**5.23.1.28 read\_met\_geopot()** void read\_met\_geopot (  
     ctl\_t \* ctl,  
     met\_t \* met )

Calculate geopotential heights.

Definition at line 2935 of file libtrac.c.

```

02937     {
02938
02939     const int dx = ctl->met_geopot_sx, dy = ctl->met_geopot_sy;
02940
02941     static float help[EX][EY][EP], w, wsum;
02942
02943     double h2os, logp[EP], ts, z0;
02944
02945     int ip, ip0, ix, ix2, ix3, iy, iy2;
02946
02947     /* Set timer... */
02948     SELECT_TIMER("READ_MET_GEOPOT", NVTX_READ);
02949
02950     /* Calculate log pressure... */
02951     for (ip = 0; ip < met->np; ip++)
02952         logp[ip] = log(met->p[ip]);
02953
02954     /* Initialize geopotential heights... */
02955     #pragma omp parallel for default(shared) private(ix,iy,ip)
02956     for (ix = 0; ix < met->nx; ix++)
02957         for (iy = 0; iy < met->ny; iy++)
02958             for (ip = 0; ip < met->np; ip++)
02959                 met->z[ix][iy][ip] = GSL_NAN;
02960 }
```

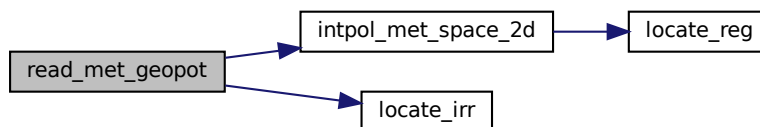


```

02961  /* Apply hydrostatic equation to calculate geopotential heights... */
02962 #pragma omp parallel for default(shared) private(ix,iy,z0,ip0,ts,ip)
02963 for (ix = 0; ix < met->nx; ix++)
02964     for (iy = 0; iy < met->ny; iy++) {
02965
02966         /* Get surface height... */
02967         INTPOL_INIT;
02968         intpol_met_space_2d(met, met->z0, met->lon[ix], met->lat[iy], &z0, ci,
02969                             cw, 1);
02970
02971         /* Find surface pressure level index... */
02972         ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
02973
02974         /* Get temperature and water vapor vmr at the surface... */
02975         ts =
02976             LIN(met->p[ip0], met->t[ix][iy][ip0], met->p[ip0 + 1],
02977                 met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
02978         h2os =
02979             LIN(met->p[ip0], met->h2o[ix][iy][ip0], met->p[ip0 + 1],
02980                 met->h2o[ix][iy][ip0 + 1], met->ps[ix][iy]);
02981
02982         /* Upper part of profile... */
02983         met->z[ix][iy][ip0 + 1]
02984             = (float) (z0 +
02985                     ZDIFF(log(met->ps[ix][iy]), ts, h2os, logp[ip0 + 1],
02986                           met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]));
02987         for (ip = ip0 + 2; ip < met->np; ip++)
02988             met->z[ix][iy][ip]
02989                 = (float) (met->z[ix][iy][ip - 1] +
02990                           ZDIFF(logp[ip - 1], met->t[ix][iy][ip - 1],
02991                                 met->h2o[ix][iy][ip - 1], logp[ip],
02992                                 met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
02993
02994         /* Lower part of profile... */
02995         met->z[ix][iy][ip0]
02996             = (float) (z0 +
02997                     ZDIFF(log(met->ps[ix][iy]), ts, h2os, logp[ip0],
02998                           met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]));
02999         for (ip = ip0 - 1; ip >= 0; ip--)
03000             met->z[ix][iy][ip]
03001                 = (float) (met->z[ix][iy][ip + 1] +
03002                           ZDIFF(logp[ip + 1], met->t[ix][iy][ip + 1],
03003                                 met->h2o[ix][iy][ip + 1], logp[ip],
03004                                 met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
03005     }
03006
03007     /* Horizontal smoothing... */
03008 #pragma omp parallel for default(shared) private(ix,iy,ip,ix2,ix3,iy2,w,wsum)
03009     for (ix = 0; ix < met->nx; ix++)
03010         for (iy = 0; iy < met->ny; iy++)
03011             for (ip = 0; ip < met->np; ip++) {
03012                 wsum = 0;
03013                 help[ix][iy][ip] = 0;
03014                 for (ix2 = ix - dx + 1; ix2 <= ix + dx - 1; ix2++) {
03015                     ix3 = ix2;
03016                     if (ix3 < 0)
03017                         ix3 += met->nx;
03018                     else if (ix3 >= met->nx)
03019                         ix3 -= met->nx;
03020                     for (iy2 = GSL_MAX(iy - dy + 1, 0);
03021                         iy2 <= GSL_MIN(iy + dy - 1, met->ny - 1); iy2++)
03022                         if (isfinite(met->z[ix3][iy2][ip])) {
03023                             w = (1.0f - (float) abs(ix - ix2) / (float) dx)
03024                                 * (1.0f - (float) abs(iy - iy2) / (float) dy);
03025                             help[ix][iy][ip] += w * met->z[ix3][iy2][ip];
03026                             wsum += w;
03027                         }
03028                 }
03029                 if (wsum > 0)
03030                     help[ix][iy][ip] /= wsum;
03031                 else
03032                     help[ix][iy][ip] = GSL_NAN;
03033             }
03034
03035     /* Copy data... */
03036 #pragma omp parallel for default(shared) private(ix,iy,ip)
03037     for (ix = 0; ix < met->nx; ix++)
03038         for (iy = 0; iy < met->ny; iy++)
03039             for (ip = 0; ip < met->np; ip++)
03040                 met->z[ix][iy][ip] = help[ix][iy][ip];
03041 }

```

Here is the call graph for this function:



**5.23.1.29 read\_met\_grid()** void read\_met\_grid (  
     char \* filename,  
     int ncid,  
     ctl\_t \* ctl,  
     met\_t \* met )

Read coordinates of meteorological data.

Definition at line 3045 of file libtrac.c.

```

03049     {
03050
03051     char levname[LEN], tstr[10];
03052
03053     int dimid, ip, varid, year, mon, day, hour;
03054
03055     size_t np, nx, ny;
03056
03057     /* Set timer... */
03058     SELECT_TIMER("READ_MET_GRID", NVTX_READ);
03059
03060     /* Get time from filename... */
03061     sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
03062     year = atoi(tstr);
03063     sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
03064     mon = atoi(tstr);
03065     sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
03066     day = atoi(tstr);
03067     sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
03068     hour = atoi(tstr);
03069     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
03070
03071     /* Get grid dimensions... */
03072     NC(nc_inq_dimid(ncid, "lon", &dimid));
03073     NC(nc_inq_dimlen(ncid, dimid, &nx));
03074     if (nx < 2 || nx > EX)
03075         ERRMSG("Number of longitudes out of range!");
03076
03077     NC(nc_inq_dimid(ncid, "lat", &dimid));
03078     NC(nc_inq_dimlen(ncid, dimid, &ny));
03079     if (ny < 2 || ny > EY)
03080         ERRMSG("Number of latitudes out of range!");
03081
03082     sprintf(levname, "lev");
03083     NC(nc_inq_dimid(ncid, levname, &dimid));
03084     NC(nc_inq_dimlen(ncid, dimid, &np));
03085     if (np == 1) {
03086         sprintf(levname, "lev_2");
03087         if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
03088             sprintf(levname, "plev");
03089             nc_inq_dimid(ncid, levname, &dimid);
03090         }
03091         NC(nc_inq_dimlen(ncid, dimid, &np));
03092     }
03093     if (np < 2 || np > EP)
03094         ERRMSG("Number of levels out of range!");
03095
03096     /* Store dimensions... */

```

```

03097 met->np = (int) np;
03098 met->nx = (int) nx;
03099 met->ny = (int) ny;
03100
03101 /* Read longitudes and latitudes... */
03102 NC(nc_inq_varid(ncid, "lon", &varid));
03103 NC(nc_get_var_double(ncid, varid, met->lon));
03104 NC(nc_inq_varid(ncid, "lat", &varid));
03105 NC(nc_get_var_double(ncid, varid, met->lat));
03106
03107 /* Read pressure levels... */
03108 if (ctl->met_np <= 0) {
03109     NC(nc_inq_varid(ncid, levname, &varid));
03110     NC(nc_get_var_double(ncid, varid, met->p));
03111     for (ip = 0; ip < met->np; ip++)
03112         met->p[ip] /= 100.;
03113 }
03114
03115 /* Set pressure levels... */
03116 else {
03117     met->np = ctl->met_np;
03118     for (ip = 0; ip < met->np; ip++)
03119         met->p[ip] = ctl->met_p[ip];
03120 }
03121
03122 /* Check ordering of pressure levels... */
03123 for (ip = 1; ip < met->np; ip++)
03124     if (met->p[ip - 1] < met->p[ip])
03125         ERRMSG("Pressure levels must be descending!");
03126 }

```

Here is the call graph for this function:



**5.23.1.30 read\_met\_help\_3d()** int read\_met\_help\_3d (

```

    int ncid,
    char * varname,
    char * varname2,
    met_t * met,
    float dest[EX][EY][EP],
    float scl )

```

Read and convert 3D variable from meteorological data file.

Definition at line 3130 of file [libtrac.c](#).

```

03136 {
03137
03138     float *help;
03139
03140     int ip, ix, iy, varid;
03141
03142     /* Check if variable exists... */
03143     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03144         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
03145             return 0;
03146
03147     /* Allocate... */
03148     ALLOC(help, float,
03149           EX * EY * EP);
03150

```

```

03151  /* Read data... */
03152  NC(nc_get_var_float(ncid, varid, help));
03153
03154  /* Copy and check data... */
03155  #pragma omp parallel for default(shared) private(ix,iy,ip)
03156  for (ix = 0; ix < met->nx; ix++)
03157      for (iy = 0; iy < met->ny; iy++)
03158          for (ip = 0; ip < met->np; ip++) {
03159              dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
03160              if (fabsf(dest[ix][iy][ip]) < 1e14f)
03161                  dest[ix][iy][ip] *= scl;
03162              else
03163                  dest[ix][iy][ip] = GSL_NAN;
03164          }
03165
03166  /* Free... */
03167  free(help);
03168
03169  /* Return... */
03170  return 1;
03171 }

```

**5.23.1.31 read\_met\_help\_2d()** int read\_met\_help\_2d (

```

    int ncid,
    char * varname,
    char * varname2,
    met_t * met,
    float dest[EX][EY],
    float scl )

```

Read and convert 2D variable from meteorological data file.

Definition at line 3175 of file libtrac.c.

```

03181  {
03182
03183  float *help;
03184
03185  int ix, iy, varid;
03186
03187  /* Check if variable exists... */
03188  if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03189      if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
03190          return 0;
03191
03192  /* Allocate... */
03193  ALLOC(help, float,
03194        EX * EY);
03195
03196  /* Read data... */
03197  NC(nc_get_var_float(ncid, varid, help));
03198
03199  /* Copy and check data... */
03200  #pragma omp parallel for default(shared) private(ix,iy)
03201  for (ix = 0; ix < met->nx; ix++)
03202      for (iy = 0; iy < met->ny; iy++) {
03203          dest[ix][iy] = help[iy * met->nx + ix];
03204          if (fabsf(dest[ix][iy]) < 1e14f)
03205              dest[ix][iy] *= scl;
03206          else
03207              dest[ix][iy] = GSL_NAN;
03208      }
03209
03210  /* Free... */
03211  free(help);
03212
03213  /* Return... */
03214  return 1;
03215 }

```

**5.23.1.32 read\_met\_levels()** void read\_met\_levels (

```

    int ncid,
    ctl_t * ctl,
    met_t * met )

```

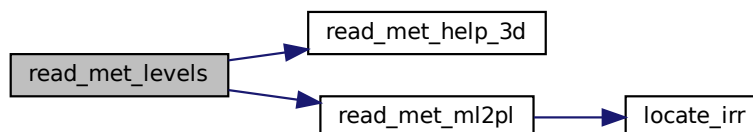
Definition at line 3219 of file libtrac.c.

```

03222     {
03223
03224     /* Set timer... */
03225     SELECT_TIMER("READ_MET_LEVELS", NVTX_READ);
03226
03227     /* Read meteorological data... */
03228     if (!read_met_help_3d(ncid, "t", "T", met, met->t, 1.0))
03229         ERRMSG("Cannot read temperature!");
03230     if (!read_met_help_3d(ncid, "u", "U", met, met->u, 1.0))
03231         ERRMSG("Cannot read zonal wind!");
03232     if (!read_met_help_3d(ncid, "v", "V", met, met->v, 1.0))
03233         ERRMSG("Cannot read meridional wind!");
03234     if (!read_met_help_3d(ncid, "w", "W", met, met->w, 0.01f))
03235         WARN("Cannot read vertical velocity");
03236     if (!read_met_help_3d(ncid, "q", "Q", met, met->h2o, (float) (MA / MH2O)))
03237         WARN("Cannot read specific humidity!");
03238     if (!read_met_help_3d(ncid, "o3", "O3", met, met->o3, (float) (MA / MO3)))
03239         WARN("Cannot read ozone data!");
03240     if (!read_met_help_3d(ncid, "clwc", "CLWC", met, met->lwc, 1.0))
03241         WARN("Cannot read cloud liquid water content!");
03242     if (!read_met_help_3d(ncid, "ciwc", "CIWC", met, met->iwc, 1.0))
03243         WARN("Cannot read cloud ice water content!");
03244
03245     /* Transfer from model levels to pressure levels... */
03246     if (ctl->met_np > 0) {
03247
03248         /* Read pressure on model levels... */
03249         if (!read_met_help_3d(ncid, "pl", "PL", met, met->pl, 0.01f))
03250             ERRMSG("Cannot read pressure on model levels!");
03251
03252         /* Vertical interpolation from model to pressure levels... */
03253         read_met_ml2pl(ctl, met, met->t);
03254         read_met_ml2pl(ctl, met, met->u);
03255         read_met_ml2pl(ctl, met, met->v);
03256         read_met_ml2pl(ctl, met, met->w);
03257         read_met_ml2pl(ctl, met, met->h2o);
03258         read_met_ml2pl(ctl, met, met->o3);
03259         read_met_ml2pl(ctl, met, met->lwc);
03260         read_met_ml2pl(ctl, met, met->iwc);
03261     }
03262 }

```

Here is the call graph for this function:



**5.23.1.33 read\_met\_ml2pl()** void read\_met\_ml2pl (

```

    ctl_t * ctl,
    met_t * met,
    float var[EX][EY][EP] )

```

Convert meteorological data from model levels to pressure levels.

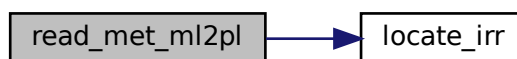
Definition at line 3266 of file [libtrac.c](#).

```

03269     {
03270
03271     double aux[EP], p[EP], pt;
03272
03273     int ip, ip2, ix, iy;
03274
03275     /* Set timer... */
03276     SELECT_TIMER("READ_MET_ML2PL", NVTX_READ);
03277
03278     /* Loop over columns... */
03279     #pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
03280     for (ix = 0; ix < met->nx; ix++)
03281         for (iy = 0; iy < met->ny; iy++) {
03282
03283             /* Copy pressure profile... */
03284             for (ip = 0; ip < met->np; ip++)
03285                 p[ip] = met->p1[ix][iy][ip];
03286
03287             /* Interpolate... */
03288             for (ip = 0; ip < ctl->met_np; ip++) {
03289                 pt = ctl->met_p[ip];
03290                 if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
03291                     pt = p[0];
03292                 else if ((pt > p[met->np - 1] && p[1] > p[0])
03293                        || (pt < p[met->np - 1] && p[1] < p[0]))
03294                     pt = p[met->np - 1];
03295                 ip2 = locate_irr(p, met->np, pt);
03296                 aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
03297                             p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
03298             }
03299
03300             /* Copy data... */
03301             for (ip = 0; ip < ctl->met_np; ip++)
03302                 var[ix][iy][ip] = (float) aux[ip];
03303         }
03304     }

```

Here is the call graph for this function:



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

Create meteorological data with periodic boundary conditions.

Definition at line 3308 of file [libtrac.c](#).

```

03309     {
03310
03311     /* Set timer... */
03312     SELECT_TIMER("READ_MET_PERIODIC", NVTX_READ);
03313
03314     /* Check longitudes... */
03315     if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
03316              + met->lon[1] - met->lon[0] - 360) < 0.01))
03317         return;
03318
03319     /* Increase longitude counter... */
03320     if ((++met->nx) > EX)
03321         ERRMSG("Cannot create periodic boundary conditions!");
03322
03323     /* Set longitude... */

```

```

03324   met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->lon[0];
03325
03326   /* Loop over latitudes and pressure levels... */
03327 #pragma omp parallel for default(shared)
03328   for (int iy = 0; iy < met->ny; iy++) {
03329     met->ps[met->nx - 1][iy] = met->ps[0][iy];
03330     met->zs[met->nx - 1][iy] = met->zs[0][iy];
03331     for (int ip = 0; ip < met->np; ip++) {
03332       met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
03333       met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
03334       met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
03335       met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
03336       met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
03337       met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
03338       met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
03339       met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
03340     }
03341   }
03342 }

```

**5.23.1.35 read\_met\_pv()** void read\_met\_pv (  
     met\_t \* met )

Calculate potential vorticity.

Definition at line 3346 of file libtrac.c.

```

03347   {
03348
03349     double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
03350           dtdp, dudp, dvdp, latr, vort, pows[EP];
03351
03352     int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
03353
03354     /* Set timer... */
03355     SELECT_TIMER("READ_MET_PV", NVTX_READ);
03356
03357     /* Set powers... */
03358     for (ip = 0; ip < met->np; ip++)
03359       pows[ip] = pow(1000. / met->p[ip], 0.286);
03360
03361     /* Loop over grid points... */
03362 #pragma omp parallel for default(shared)
03363     private(ix,ix0,ix1,iy,iy0,iy1,latr,dx,dy,c0,c1,cr,vort,ip,ip0,ip1,dp0,dp1,denom,dtdx,dvdx,dtdy,dudy,dtdp,dudp,dvdp)
03364     for (ix = 0; ix < met->nx; ix++) {
03365
03366       /* Set indices... */
03367       ix0 = GSL_MAX(ix - 1, 0);
03368       ix1 = GSL_MIN(ix + 1, met->nx - 1);
03369
03370       /* Loop over grid points... */
03371       for (iy = 0; iy < met->ny; iy++) {
03372
03373         /* Set indices... */
03374         iy0 = GSL_MAX(iy - 1, 0);
03375         iy1 = GSL_MIN(iy + 1, met->ny - 1);
03376
03377         /* Set auxiliary variables... */
03378         latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
03379         dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
03380         dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
03381         c0 = cos(met->lat[iy0] / 180. * M_PI);
03382         c1 = cos(met->lat[iy1] / 180. * M_PI);
03383         cr = cos(latr / 180. * M_PI);
03384         vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
03385
03386         /* Loop over grid points... */
03387         for (ip = 0; ip < met->np; ip++) {
03388
03389           /* Get gradients in longitude... */
03390           dtdx = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
03391           dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
03392
03393           /* Get gradients in latitude... */
03394           dtdy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
03395           dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
03396
03397           /* Set indices... */
03398           ip0 = GSL_MAX(ip - 1, 0);
03399           ip1 = GSL_MIN(ip + 1, met->np - 1);

```

```

03399
03400      /* Get gradients in pressure... */
03401      dp0 = 100. * (met->p[ip] - met->p[ip0]);
03402      dp1 = 100. * (met->p[ip1] - met->p[ip]);
03403      if (ip != ip0 && ip != ip1) {
03404          denom = dp0 * dp1 * (dp0 + dp1);
03405          dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
03406                - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
03407                + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
03408                / denom;
03409          dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
03410                - dp1 * dp1 * met->u[ix][iy][ip0]
03411                + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
03412                / denom;
03413          dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
03414                - dp1 * dp1 * met->v[ix][iy][ip0]
03415                + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
03416                / denom;
03417      } else {
03418          denom = dp0 + dp1;
03419          dtdp =
03420              (met->t[ix][iy][ip1] * pows[ip1] -
03421               met->t[ix][iy][ip0] * pows[ip0]) / denom;
03422          dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
03423          dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
03424      }
03425
03426      /* Calculate PV... */
03427      met->pv[ix][iy][ip] = (float)
03428          (1e6 * G0 *
03429           (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
03430  }
03431  }
03432  }
03433
03434      /* Fix for polar regions... */
03435      #pragma omp parallel for default(shared) private(ix,ip)
03436      for (ix = 0; ix < met->nx; ix++)
03437          for (ip = 0; ip < met->np; ip++) {
03438              met->pv[ix][0][ip]
03439                  = met->pv[ix][1][ip]
03440                  = met->pv[ix][2][ip];
03441              met->pv[ix][met->ny - 1][ip]
03442                  = met->pv[ix][met->ny - 2][ip]
03443                  = met->pv[ix][met->ny - 3][ip];
03444          }
03445  }

```

**5.23.1.36 read\_met\_sample()** void read\_met\_sample (  
     ctl\_t \* ctl,  
     met\_t \* met )

Downsampling of meteorological data.

Definition at line 3449 of file libtrac.c.

```

03451      {
03452
03453          met_t *help;
03454
03455          float w, wsum;
03456
03457          int ip, ip2, ix, ix2, ix3, iy, iy2;
03458
03459          /* Check parameters... */
03460          if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
03461              && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
03462              return;
03463
03464          /* Set timer... */
03465          SELECT_TIMER("READ_MET_SAMPLE", NVTX_READ);
03466
03467          /* Allocate... */
03468          ALLOC(help, met_t, 1);
03469
03470          /* Copy data... */
03471          help->nx = met->nx;
03472          help->ny = met->ny;
03473          help->np = met->np;

```



```

03474 memcpy(help->lon, met->lon, sizeof(met->lon));
03475 memcpy(help->lat, met->lat, sizeof(met->lat));
03476 memcpy(help->p, met->p, sizeof(met->p));
03477
03478 /* Smoothing... */
03479 for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
03480     for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
03481         for (ip = 0; ip < met->np; ip += ctl->met_dp) {
03482             help->ps[ix][iy] = 0;
03483             help->zs[ix][iy] = 0;
03484             help->t[ix][iy][ip] = 0;
03485             help->u[ix][iy][ip] = 0;
03486             help->v[ix][iy][ip] = 0;
03487             help->w[ix][iy][ip] = 0;
03488             help->h2o[ix][iy][ip] = 0;
03489             help->o3[ix][iy][ip] = 0;
03490             help->lwc[ix][iy][ip] = 0;
03491             help->iwc[ix][iy][ip] = 0;
03492             wsum = 0;
03493             for (ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1; ix2++) {
03494                 ix3 = ix2;
03495                 if (ix3 < 0)
03496                     ix3 += met->nx;
03497                 else if (ix3 >= met->nx)
03498                     ix3 -= met->nx;
03499
03500                 for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
03501                     iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
03502                     for (ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
03503                         ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
03504                         w = (1.0f - (float) abs(ix - ix2) / (float) ctl->met_sx)
03505                             * (1.0f - (float) abs(iy - iy2) / (float) ctl->met_sy)
03506                             * (1.0f - (float) abs(ip - ip2) / (float) ctl->met_sp);
03507                         help->ps[ix][iy] += w * met->ps[ix3][iy2];
03508                         help->zs[ix][iy] += w * met->zs[ix3][iy2];
03509                         help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
03510                         help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
03511                         help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
03512                         help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
03513                         help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
03514                         help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
03515                         help->lwc[ix][iy][ip] += w * met->lwc[ix3][iy2][ip2];
03516                         help->iwc[ix][iy][ip] += w * met->iwc[ix3][iy2][ip2];
03517                         wsum += w;
03518                     }
03519             }
03520             help->ps[ix][iy] /= wsum;
03521             help->zs[ix][iy] /= wsum;
03522             help->t[ix][iy][ip] /= wsum;
03523             help->u[ix][iy][ip] /= wsum;
03524             help->v[ix][iy][ip] /= wsum;
03525             help->w[ix][iy][ip] /= wsum;
03526             help->h2o[ix][iy][ip] /= wsum;
03527             help->o3[ix][iy][ip] /= wsum;
03528             help->lwc[ix][iy][ip] /= wsum;
03529             help->iwc[ix][iy][ip] /= wsum;
03530         }
03531     }
03532 }
03533
03534 /* Downsampling... */
03535 met->nx = 0;
03536 for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
03537     met->lon[met->nx] = help->lon[ix];
03538     met->ny = 0;
03539     for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
03540         met->lat[met->ny] = help->lat[iy];
03541         met->ps[met->nx][met->ny] = help->ps[ix][iy];
03542         met->zs[met->nx][met->ny] = help->zs[ix][iy];
03543         met->np = 0;
03544         for (ip = 0; ip < help->np; ip += ctl->met_dp) {
03545             met->p[met->np] = help->p[ip];
03546             met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
03547             met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
03548             met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
03549             met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
03550             met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
03551             met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
03552             met->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];
03553             met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
03554             met->np++;
03555         }
03556         met->ny++;
03557     }
03558     met->nx++;
03559 }
03560

```

```

03561  /* Free... */
03562  free(help);
03563 }

```

**5.23.1.37 read\_met\_surface()** void read\_met\_surface (

```

    int ncid,
    met_t * met )

```

Read surface data.

Definition at line 3567 of file libtrac.c.

```

03569 {
03570
03571     int ix, iy;
03572
03573     /* Set timer... */
03574     SELECT_TIMER("READ_MET_SURFACE", NVTX_READ);
03575
03576     /* Read surface pressure... */
03577     if (!read_met_help_2d(ncid, "ps", "PS", met, met->ps, 0.01f)) {
03578         if (!read_met_help_2d(ncid, "lnsp", "LNSP", met, met->ps, 1.0)) {
03579             ERRMSG("Cannot not read surface pressure data!");
03580             for (ix = 0; ix < met->nx; ix++)
03581                 for (iy = 0; iy < met->ny; iy++)
03582                     met->ps[ix][iy] = (float) met->p[0];
03583         } else {
03584             for (iy = 0; iy < met->ny; iy++)
03585                 for (ix = 0; ix < met->nx; ix++)
03586                     met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
03587         }
03588     }
03589
03590     /* Read geopotential height at the surface... */
03591     if (!read_met_help_2d
03592         (ncid, "z", "Z", met, met->zs, (float) (1. / (1000. * G0))))
03593         if (!read_met_help_2d
03594             (ncid, "zm", "ZM", met, met->zs, (float) (1. / 1000.)))
03595             ERRMSG("Cannot read surface geopotential height!");
03596
03597     /* Read temperature at the surface... */
03598     if (!read_met_help_2d(ncid, "t2m", "T2M", met, met->ts, 1.0))
03599         WARN("Cannot read surface temperature!");
03600
03601     /* Read zonal wind at the surface... */
03602     if (!read_met_help_2d(ncid, "u10m", "U10M", met, met->us, 1.0))
03603         WARN("Cannot read surface zonal wind!");
03604
03605     /* Read meridional wind at the surface... */
03606     if (!read_met_help_2d(ncid, "v10m", "V10M", met, met->vs, 1.0))
03607         WARN("Cannot read surface meridional wind!");
03608 }

```

Here is the call graph for this function:



**5.23.1.38 read\_met\_tropo()** void read\_met\_tropo (  
     ctl\_t \* ctl,  
     met\_t \* met )

Calculate tropopause data.

Definition at line 3612 of file libtrac.c.

```

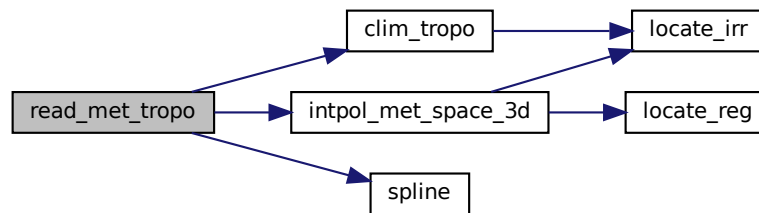
03614     {
03615
03616     double h2ot, p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
03617         th2[200], tt, z[EP], z2[200], zt;
03618
03619     int found, ix, iy, iz, iz2;
03620
03621     /* Set timer... */
03622     SELECT_TIMER("READ_MET_TROPO", NVTX_READ);
03623
03624     /* Get altitude and pressure profiles... */
03625     for (iz = 0; iz < met->np; iz++)
03626         z[iz] = Z(met->p[iz]);
03627     for (iz = 0; iz <= 190; iz++) {
03628         z2[iz] = 4.5 + 0.1 * iz;
03629         p2[iz] = P(z2[iz]);
03630     }
03631
03632     /* Do not calculate tropopause... */
03633     if (ctl->met_tropo == 0)
03634         for (ix = 0; ix < met->nx; ix++)
03635             for (iy = 0; iy < met->ny; iy++)
03636                 met->pt[ix][iy] = GSL_NAN;
03637
03638     /* Use tropopause climatology... */
03639     else if (ctl->met_tropo == 1) {
03640 #pragma omp parallel for default(shared) private(ix,iy)
03641         for (ix = 0; ix < met->nx; ix++)
03642             for (iy = 0; iy < met->ny; iy++)
03643                 met->pt[ix][iy] = (float) clim_tropo(met->time, met->lat[iy]);
03644     }
03645
03646     /* Use cold point... */
03647     else if (ctl->met_tropo == 2) {
03648
03649         /* Loop over grid points... */
03650 #pragma omp parallel for default(shared) private(ix,iy,iz,t,t2)
03651         for (ix = 0; ix < met->nx; ix++)
03652             for (iy = 0; iy < met->ny; iy++) {
03653
03654                 /* Interpolate temperature profile... */
03655                 for (iz = 0; iz < met->np; iz++)
03656                     t[iz] = met->t[ix][iy][iz];
03657                 spline(z, t, met->np, z2, t2, 171);
03658
03659                 /* Find minimum... */
03660                 iz = (int) gsl_stats_min_index(t2, 1, 171);
03661                 if (iz > 0 && iz < 170)
03662                     met->pt[ix][iy] = (float) p2[iz];
03663                 else
03664                     met->pt[ix][iy] = GSL_NAN;
03665             }
03666     }
03667
03668     /* Use WMO definition... */
03669     else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
03670
03671         /* Loop over grid points... */
03672 #pragma omp parallel for default(shared) private(ix,iy,iz,iz2,t,t2,found)
03673         for (ix = 0; ix < met->nx; ix++)
03674             for (iy = 0; iy < met->ny; iy++) {
03675
03676                 /* Interpolate temperature profile... */
03677                 for (iz = 0; iz < met->np; iz++)
03678                     t[iz] = met->t[ix][iy][iz];
03679                 spline(z, t, met->np, z2, t2, 191);
03680
03681                 /* Find 1st tropopause... */
03682                 met->pt[ix][iy] = GSL_NAN;
03683                 for (iz = 0; iz <= 170; iz++) {
03684                     found = 1;
03685                     for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
03686                         if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) > 2.0) {
03687                             found = 0;
03688                             break;
03689                         }
03690                     if (found) {
```

```

03691         if (iz > 0 && iz < 170)
03692             met->pt[ix][iy] = (float) p2[iz];
03693         break;
03694     }
03695 }
03696
03697 /* Find 2nd tropopause... */
03698 if (ctl->met_tropo == 4) {
03699     met->pt[ix][iy] = GSL_NAN;
03700     for (; iz <= 170; iz++) {
03701         found = 1;
03702         for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
03703             if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) < 3.0) {
03704                 found = 0;
03705                 break;
03706             }
03707         if (found)
03708             break;
03709     }
03710     for (; iz <= 170; iz++) {
03711         found = 1;
03712         for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
03713             if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) > 2.0) {
03714                 found = 0;
03715                 break;
03716             }
03717         if (found) {
03718             if (iz > 0 && iz < 170)
03719                 met->pt[ix][iy] = (float) p2[iz];
03720             break;
03721         }
03722     }
03723 }
03724 }
03725 }
03726
03727 /* Use dynamical tropopause... */
03728 else if (ctl->met_tropo == 5) {
03729
03730     /* Loop over grid points... */
03731     #pragma omp parallel for default(shared) private(ix,iy,iz,pv,pv2,th,th2)
03732     for (ix = 0; ix < met->nx; ix++)
03733         for (iy = 0; iy < met->ny; iy++) {
03734
03735             /* Interpolate potential vorticity profile... */
03736             for (iz = 0; iz < met->np; iz++)
03737                 pv[iz] = met->pv[ix][iy][iz];
03738             spline(z, pv, met->np, z2, pv2, 171);
03739
03740             /* Interpolate potential temperature profile... */
03741             for (iz = 0; iz < met->np; iz++)
03742                 th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
03743             spline(z, th, met->np, z2, th2, 171);
03744
03745             /* Find dynamical tropopause 3.5 PVU + 380 K */
03746             met->pt[ix][iy] = GSL_NAN;
03747             for (iz = 0; iz <= 170; iz++)
03748                 if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
03749                     if (iz > 0 && iz < 170)
03750                         met->pt[ix][iy] = (float) p2[iz];
03751                     break;
03752                 }
03753         }
03754     }
03755
03756     else
03757         ERRMSG("Cannot calculate tropopause!");
03758
03759     /* Interpolate temperature, geopotential height, and water vapor vmr... */
03760     #pragma omp parallel for default(shared) private(ix,iy,tt,zt,h2ot)
03761     for (ix = 0; ix < met->nx; ix++)
03762         for (iy = 0; iy < met->ny; iy++) {
03763             INTPOL_INIT;
03764             intpol_met_space_3d(met, met->t, met->pt[ix][iy], met->lon[ix],
03765                                 met->lat[iy], &tt, ci, cw, 1);
03766             intpol_met_space_3d(met, met->z, met->pt[ix][iy], met->lon[ix],
03767                                 met->lat[iy], &zt, ci, cw, 0);
03768             intpol_met_space_3d(met, met->h2o, met->pt[ix][iy], met->lon[ix],
03769                                 met->lat[iy], &h2ot, ci, cw, 0);
03770             met->tt[ix][iy] = (float) tt;
03771             met->zt[ix][iy] = (float) zt;
03772             met->h2ot[ix][iy] = (float) h2ot;
03773         }
03774 }

```

Here is the call graph for this function:



**5.23.1.39 scan\_ctl()** 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 3778 of file [libtrac.c](#).

```

03785     {
03786
03787     FILE *in = NULL;
03788
03789     char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
03790         msg[2 * LEN], rvarname[LEN], rval[LEN];
03791
03792     int contain = 0, i;
03793
03794     /* Open file... */
03795     if (filename[strlen(filename) - 1] != '-')
03796         if (!(in = fopen(filename, "r")))
03797             ERRMSG("Cannot open file!");
03798
03799     /* Set full variable name... */
03800     if (arridx >= 0) {
03801         sprintf(fullname1, "%s[%d]", varname, arridx);
03802         sprintf(fullname2, "%s[*]", varname);
03803     } else {
03804         sprintf(fullname1, "%s", varname);
03805         sprintf(fullname2, "%s", varname);
03806     }
03807
03808     /* Read data... */
03809     if (in != NULL)
03810         while (fgets(line, LEN, in))
03811             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
03812                 if (strcasecmp(rvarname, fullname1) == 0 ||
03813                     strcasecmp(rvarname, fullname2) == 0) {
03814                     contain = 1;
03815                     break;
03816                 }
03817     for (i = 1; i < argc - 1; i++)
03818         if (strcasecmp(argv[i], fullname1) == 0 ||
03819             strcasecmp(argv[i], fullname2) == 0) {
03820             sprintf(rval, "%s", argv[i + 1]);
03821             contain = 1;
03822             break;
03823         }
03824

```

```

03825  /* Close file... */
03826  if (in != NULL)
03827      fclose(in);
03828
03829  /* Check for missing variables... */
03830  if (!contain) {
03831      if (strlen(defvalue) > 0)
03832          sprintf(rval, "%s", defvalue);
03833      else {
03834          sprintf(msg, "Missing variable %s!\n", fullname1);
03835          ERRMSG(msg);
03836      }
03837  }
03838
03839  /* Write info... */
03840  printf("%s = %s\n", fullname1, rval);
03841
03842  /* Return values... */
03843  if (value != NULL)
03844      sprintf(value, "%s", rval);
03845  return atof(rval);
03846 }

```

**5.23.1.40 sedi()** double sedi (  
     double *p*,  
     double *T*,  
     double *r\_p*,  
     double *rho\_p* )

Calculate sedimentation velocity.

Definition at line 3850 of file libtrac.c.

```

03854  {
03855
03856  double eta, G, K, lambda, rho, v;
03857
03858  /* Convert units... */
03859  p *= 100.;                      /* from hPa to Pa */
03860  r_p *= 1e-6;                    /* from microns to m */
03861
03862  /* Density of dry air... */
03863  rho = p / (RA * T);
03864
03865  /* Dynamic viscosity of air... */
03866  eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
03867
03868  /* Thermal velocity of an air molecule... */
03869  v = sqrt(8. * KB * T / (M_PI * 4.8096e-26));
03870
03871  /* Mean free path of an air molecule... */
03872  lambda = 2. * eta / (rho * v);
03873
03874  /* Knudsen number for air... */
03875  K = lambda / r_p;
03876
03877  /* Cunningham slip-flow correction... */
03878  G = 1. + K * (1.249 + 0.42 * exp(-0.87 / K));
03879
03880  /* Sedimentation velocity... */
03881  return 2. * SQR(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;
03882 }

```

**5.23.1.41 spline()** void spline (  
     double \* *x*,  
     double \* *y*,  
     int *n*,  
     double \* *x2*,  
     double \* *y2*,  
     int *n2* )

Spline interpolation.

Definition at line 3886 of file [libtrac.c](#).

```

03892     {
03893
03894     gsl_interp_accel *acc;
03895
03896     gsl_spline *s;
03897
03898     /* Allocate... */
03899     acc = gsl_interp_accel_alloc();
03900     s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
03901
03902     /* Interpolate profile... */
03903     gsl_spline_init(s, x, y, (size_t) n);
03904     for (int i = 0; i < n2; i++)
03905         if (x2[i] <= x[0])
03906             y2[i] = y[0];
03907         else if (x2[i] >= x[n - 1])
03908             y2[i] = y[n - 1];
03909         else
03910             y2[i] = gsl_spline_eval(s, x2[i], acc);
03911
03912     /* Free... */
03913     gsl_spline_free(s);
03914     gsl_interp_accel_free(acc);
03915 }
```

**5.23.1.42 stddev()** double stddev (  
     double \* data,  
     int n )

Calculate standard deviation.

Definition at line 3919 of file [libtrac.c](#).

```

03921     {
03922
03923     if (n <= 0)
03924         return 0;
03925
03926     double avg = 0, rms = 0;
03927
03928     for (int i = 0; i < n; ++i)
03929         avg += data[i];
03930     avg /= n;
03931
03932     for (int i = 0; i < n; ++i)
03933         rms += SQR(data[i] - avg);
03934
03935     return sqrt(rms / (n - 1));
03936 }
```

**5.23.1.43 time2jsec()** 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 3940 of file [libtrac.c](#).

```

03948     {
```

```

03949
03950     struct tm t0, t1;
03951
03952     t0.tm_year = 100;
03953     t0.tm_mon = 0;
03954     t0.tm_mday = 1;
03955     t0.tm_hour = 0;
03956     t0.tm_min = 0;
03957     t0.tm_sec = 0;
03958
03959     t1.tm_year = year - 1900;
03960     t1.tm_mon = mon - 1;
03961     t1.tm_mday = day;
03962     t1.tm_hour = hour;
03963     t1.tm_min = min;
03964     t1.tm_sec = sec;
03965
03966     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
03967 }

```

**5.23.1.44 timer()** void timer (  
     const char \* name,  
     int output )

Measure wall-clock time.

Definition at line 3971 of file libtrac.c.

```

03973     {
03974
03975     static char namelist[NTIMER][100];
03976
03977     static double runtime[NTIMER], t0, t1;
03978
03979     static int it = -1, nt;
03980
03981     /* Get time... */
03982     t1 = omp_get_wtime();
03983
03984     /* Add elapsed timer to old timer... */
03985     if (it >= 0)
03986         runtime[it] += t1 - t0;
03987
03988     /* Identify ID of new timer... */
03989     for (it = 0; it < nt; it++)
03990         if (strcascmp(name, namelist[it]) == 0)
03991             break;
03992
03993     /* Check whether this is a new timer... */
03994     if (it >= nt) {
03995         sprintf(namelist[it], "%s", name);
03996         if ((++nt) > NTIMER)
03997             ERRMSG("Too many timers!");
03998     }
03999
04000     /* Save starting time... */
04001     t0 = t1;
04002
04003     /* Report timers... */
04004     if (output) {
04005         for (int it2 = 0; it2 < nt; it2++)
04006             printf("TIMER_%s = %.3f s\n", namelist[it2], runtime[it2]);
04007         double total = 0.0;
04008         for (int it2 = 0; it2 < nt; it2++)
04009             total += runtime[it2];
04010         printf("TIMER_TOTAL = %.3f s\n", total);
04011     }
04012 }

```



```

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

```

Write atmospheric data.

Definition at line 4016 of file libtrac.c.

```

04020     {
04021
04022     FILE *in, *out;
04023
04024     char line[LEN];
04025
04026     double r, t0, t1;
04027
04028     int ip, iq, year, mon, day, hour, min, sec;
04029
04030     /* Set timer... */
04031     SELECT_TIMER("WRITE_ATM", NVTX_WRITE);
04032
04033     /* Set time interval for output... */
04034     t0 = t - 0.5 * ctl->dt_mod;
04035     t1 = t + 0.5 * ctl->dt_mod;
04036
04037     /* Write info... */
04038     printf("Write atmospheric data: %s\n", filename);
04039
04040     /* Write ASCII data... */
04041     if (ctl->atm_type == 0) {
04042
04043         /* Check if gnuplot output is requested... */
04044         if (ctl->atm_gpfile[0] != '-') {
04045
04046             /* Create gnuplot pipe... */
04047             if (!(out = popen("gnuplot", "w")))
04048                 ERRMSG("Cannot create pipe to gnuplot!");
04049
04050             /* Set plot filename... */
04051             fprintf(out, "set out \"%s.png\"\n", filename);
04052
04053             /* Set time string... */
04054             jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
04055             fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
04056                     year, mon, day, hour, min);
04057
04058             /* Dump gnuplot file to pipe... */
04059             if (!(in = fopen(ctl->atm_gpfile, "r")))
04060                 ERRMSG("Cannot open file!");
04061             while (fgets(line, LEN, in))
04062                 fprintf(out, "%s", line);
04063             fclose(in);
04064         }
04065
04066     else {
04067
04068         /* Create file... */
04069         if (!(out = fopen(filename, "w")))
04070             ERRMSG("Cannot create file!");
04071     }
04072
04073     /* Write header... */
04074     fprintf(out,
04075            "# $1 = time [s]\n"
04076            "# $2 = altitude [km]\n"
04077            "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
04078     for (iq = 0; iq < ctl->nq; iq++)
04079         fprintf(out, "# $i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
04080                ctl->qnt_unit[iq]);
04081     fprintf(out, "\n");
04082
04083     /* Write data... */
04084     for (ip = 0; ip < atm->np; ip += ctl->atm_stride) {
04085
04086         /* Check time... */
04087         if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
04088             continue;
04089
04090         /* Write output... */
04091         fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
04092                atm->lon[ip], atm->lat[ip]);
04093         for (iq = 0; iq < ctl->nq; iq++) {

```

```

04094         fprintf(out, " ");
04095         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
04096     }
04097     fprintf(out, "\n");
04098 }
04099
04100     /* Close file... */
04101     fclose(out);
04102 }
04103
04104     /* Write binary data... */
04105     else if (ctl->atm_type == 1) {
04106
04107         /* Create file... */
04108         if (!(out = fopen(filename, "w")))
04109             ERRMSG("Cannot create file!");
04110
04111         /* Write data... */
04112         FWRITE(&atm->np, int,
04113             1,
04114             out);
04115         FWRITE(atm->time, double,
04116             (size_t) atm->np,
04117             out);
04118         FWRITE(atm->p, double,
04119             (size_t) atm->np,
04120             out);
04121         FWRITE(atm->lon, double,
04122             (size_t) atm->np,
04123             out);
04124         FWRITE(atm->lat, double,
04125             (size_t) atm->np,
04126             out);
04127         for (iq = 0; iq < ctl->nq; iq++)
04128             FWRITE(atm->q[iq], double,
04129                 (size_t) atm->np,
04130                 out);
04131
04132         /* Close file... */
04133         fclose(out);
04134     }
04135
04136     /* Error... */
04137     else
04138         ERRMSG("Atmospheric data type not supported!");
04139 }

```

Here is the call graph for this function:



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

Write CSI data.

Definition at line 4143 of file libtrac.c.

```

04147     {
04148

```

```

04149 static FILE *in, *out;
04150
04151 static char line[LEN];
04152
04153 static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ], rt, rt_old = -1e99,
04154 rz, rlon, rlat, robs, t0, t1, area[GY], dlon, dlat, dz, lat,
04155 x[1000000], y[1000000], work[2000000];
04156
04157 static int obscount[GX][GY][GZ], ct, cx, cy, cz, ip, ix, iy, iz, n;
04158
04159 /* Set timer... */
04160 SELECT_TIMER("WRITE_CSI", NVTX_WRITE);
04161
04162 /* Init... */
04163 if (t == ctl->t_start) {
04164
04165     /* Check quantity index for mass... */
04166     if (ctl->qnt_m < 0)
04167         ERRMSG("Need quantity mass!");
04168
04169     /* Open observation data file... */
04170     printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
04171     if (!(in = fopen(ctl->csi_obsfile, "r")))
04172         ERRMSG("Cannot open file!");
04173
04174     /* Create new file... */
04175     printf("Write CSI data: %s\n", filename);
04176     if (!(out = fopen(filename, "w")))
04177         ERRMSG("Cannot create file!");
04178
04179     /* Write header... */
04180     fprintf(out,
04181         "# $1 = time [s]\n"
04182         "# $2 = number of hits (cx)\n"
04183         "# $3 = number of misses (cy)\n"
04184         "# $4 = number of false alarms (cz)\n"
04185         "# $5 = number of observations (cx + cy)\n"
04186         "# $6 = number of forecasts (cx + cz)\n"
04187         "# $7 = bias (ratio of forecasts and observations) [%%]\n"
04188         "# $8 = probability of detection (POD) [%%]\n"
04189         "# $9 = false alarm rate (FAR) [%%]\n"
04190         "# $10 = critical success index (CSI) [%%]\n");
04191     fprintf(out,
04192         "# $11 = hits associated with random chance\n"
04193         "# $12 = equitable threat score (ETS) [%%]\n"
04194         "# $13 = Pearson linear correlation coefficient\n"
04195         "# $14 = Spearman rank-order correlation coefficient\n"
04196         "# $15 = column density mean error (F - O) [kg/m^2]\n"
04197         "# $16 = column density root mean square error (RMSE) [kg/m^2]\n"
04198         "# $17 = column density mean absolute error [kg/m^2]\n"
04199         "# $18 = number of data points\n\n");
04200
04201     /* Set grid box size... */
04202     dz = (ctl->csi_z1 - ctl->csi_z0) / ctl->csi_nz;
04203     dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
04204     dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
04205
04206     /* Set horizontal coordinates... */
04207     for (iy = 0; iy < ctl->csi_ny; iy++) {
04208         lat = ctl->csi_lat0 + dlat * (iy + 0.5);
04209         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.) * cos(lat * M_PI / 180.);
04210     }
04211 }
04212
04213 /* Set time interval... */
04214 t0 = t - 0.5 * ctl->dt_mod;
04215 t1 = t + 0.5 * ctl->dt_mod;
04216
04217 /* Initialize grid cells... */
04218 #pragma omp parallel for default(shared) private(ix,iy,iz)
04219 for (ix = 0; ix < ctl->csi_nx; ix++)
04220     for (iy = 0; iy < ctl->csi_ny; iy++)
04221         for (iz = 0; iz < ctl->csi_nz; iz++)
04222             modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
04223
04224 /* Read observation data... */
04225 while (fgets(line, LEN, in)) {
04226
04227     /* Read data... */
04228     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
04229         5)
04230         continue;
04231
04232     /* Check time... */
04233     if (rt < t0)
04234         continue;
04235     if (rt > t1)

```

```

04236     break;
04237     if (rt < rt_old)
04238         ERRMSG("Time must be ascending!");
04239     rt_old = rt;
04240
04241     /* Check observation data... */
04242     if (!isfinite(robs))
04243         continue;
04244
04245     /* Calculate indices... */
04246     ix = (int) ((rlon - ctl->csi_lon0) / dlon);
04247     iy = (int) ((rlat - ctl->csi_lat0) / dlat);
04248     iz = (int) ((rz - ctl->csi_z0) / dz);
04249
04250     /* Check indices... */
04251     if (ix < 0 || ix >= ctl->csi_nx ||
04252         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
04253         continue;
04254
04255     /* Get mean observation index... */
04256     obsmean[ix][iy][iz] += robs;
04257     obscount[ix][iy][iz]++;
04258 }
04259
04260 /* Analyze model data... */
04261 for (ip = 0; ip < atm->np; ip++) {
04262
04263     /* Check time... */
04264     if (atm->time[ip] < t0 || atm->time[ip] > t1)
04265         continue;
04266
04267     /* Get indices... */
04268     ix = (int) ((atm->lon[ip] - ctl->csi_lon0) / dlon);
04269     iy = (int) ((atm->lat[ip] - ctl->csi_lat0) / dlat);
04270     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0) / dz);
04271
04272     /* Check indices... */
04273     if (ix < 0 || ix >= ctl->csi_nx ||
04274         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
04275         continue;
04276
04277     /* Get total mass in grid cell... */
04278     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04279 }
04280
04281 /* Analyze all grid cells... */
04282 for (ix = 0; ix < ctl->csi_nx; ix++)
04283     for (iy = 0; iy < ctl->csi_ny; iy++)
04284         for (iz = 0; iz < ctl->csi_nz; iz++) {
04285
04286             /* Calculate mean observation index... */
04287             if (obscount[ix][iy][iz] > 0)
04288                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
04289
04290             /* Calculate column density... */
04291             if (modmean[ix][iy][iz] > 0)
04292                 modmean[ix][iy][iz] /= (1e6 * area[iy]);
04293
04294             /* Calculate CSI... */
04295             if (obscount[ix][iy][iz] > 0) {
04296                 ct++;
04297                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
04298                     modmean[ix][iy][iz] >= ctl->csi_modmin)
04299                     cx++;
04300                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
04301                     modmean[ix][iy][iz] < ctl->csi_modmin)
04302                     cy++;
04303                 else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
04304                     modmean[ix][iy][iz] >= ctl->csi_modmin)
04305                     cz++;
04306             }
04307
04308             /* Save data for other verification statistics... */
04309             if (obscount[ix][iy][iz] > 0
04310                 && (obsmean[ix][iy][iz] >= ctl->csi_obsmin
04311                     || modmean[ix][iy][iz] >= ctl->csi_modmin)) {
04312                 x[n] = modmean[ix][iy][iz];
04313                 y[n] = obsmean[ix][iy][iz];
04314                 if (++n > 1000000)
04315                     ERRMSG("Too many data points to calculate statistics!");
04316             }
04317         }
04318
04319     /* Write output... */
04320     if (fmod(t, ctl->csi_dt_out) == 0) {
04321
04322         /* Calculate verification statistics

```

```

04323     (https://www.cawcr.gov.au/projects/verification/) ... */
04324     int nobs = cx + cy;
04325     int nfor = cx + cz;
04326     double bias = (nobs > 0) ? 100. * nfor / nobs : GSL_NAN;
04327     double pod = (nobs > 0) ? (100. * cx) / nobs : GSL_NAN;
04328     double far = (nfor > 0) ? (100. * cz) / nfor : GSL_NAN;
04329     double csi = (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN;
04330     double cx_rd = (ct > 0) ? (1. * nobs * nfor) / ct : GSL_NAN;
04331     double ets = (cx + cy + cz - cx_rd > 0) ?
04332         (100. * (cx - cx_rd)) / (cx + cy + cz - cx_rd) : GSL_NAN;
04333     double rho_p =
04334         (n > 0) ? gsl_stats_correlation(x, 1, y, 1, (size_t) n) : GSL_NAN;
04335     double rho_s =
04336         (n > 0) ? gsl_stats_spearman(x, 1, y, 1, (size_t) n, work) : GSL_NAN;
04337     for (int i = 0; i < n; i++)
04338         work[i] = x[i] - y[i];
04339     double mean = (n > 0) ? gsl_stats_mean(work, 1, (size_t) n) : GSL_NAN;
04340     double rmse = (n > 0) ? gsl_stats_sd_with_fixed_mean(work, 1, (size_t) n,
04341         0.0) : GSL_NAN;
04342     double absdev =
04343         (n > 0) ? gsl_stats_absdev_m(work, 1, (size_t) n, 0.0) : GSL_NAN;
04344
04345     /* Write... */
04346     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g %g %g %g %g %d\n",
04347         t, cx, cy, cz, nobs, nfor, bias, pod, far, csi, cx_rd, ets,
04348         rho_p, rho_s, mean, rmse, absdev, n);
04349
04350     /* Set counters to zero... */
04351     n = ct = cx = cy = cz = 0;
04352 }
04353
04354 /* Close file... */
04355 if (t == ctl->t_stop)
04356     fclose(out);
04357 }

```

**5.23.1.47 write\_ens()** void write\_ens (

```

    const char * filename,
    ctl_t * ctl,
    atm_t * atm,
    double t )

```

Write ensemble data.

Definition at line 4361 of file libtrac.c.

```

04365     {
04366
04367     static FILE *out;
04368
04369     static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
04370         t0, t1, x[NENS][3], xm[3];
04371
04372     static int ip, iq;
04373
04374     static size_t i, n;
04375
04376     /* Set timer... */
04377     SELECT_TIMER("WRITE_ENS", NVTX_WRITE);
04378
04379     /* Init... */
04380     if (t == ctl->t_start) {
04381
04382         /* Check quantities... */
04383         if (ctl->qnt_ens < 0)
04384             ERRMSG("Missing ensemble IDs!");
04385
04386         /* Create new file... */
04387         printf("Write ensemble data: %s\n", filename);
04388         if (!(out = fopen(filename, "w")))
04389             ERRMSG("Cannot create file!");
04390
04391         /* Write header... */
04392         fprintf(out,
04393             "# $1 = time [s]\n"
04394             "# $2 = altitude [km]\n"
04395             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
04396         for (iq = 0; iq < ctl->nq; iq++)

```

```

04397         fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
04398                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
04399     for (iq = 0; iq < ctl->nq; iq++)
04400         fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
04401                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
04402     fprintf(out, "# $%d = number of members\n", 5 + 2 * ctl->nq);
04403 }
04404
04405 /* Set time interval... */
04406 t0 = t - 0.5 * ctl->dt_mod;
04407 t1 = t + 0.5 * ctl->dt_mod;
04408
04409 /* Init... */
04410 ens = GSL_NAN;
04411 n = 0;
04412
04413 /* Loop over air parcels... */
04414 for (ip = 0; ip < atm->np; ip++) {
04415
04416     /* Check time... */
04417     if (atm->time[ip] < t0 || atm->time[ip] > t1)
04418         continue;
04419
04420     /* Check ensemble id... */
04421     if (atm->q[ctl->qnt_ens][ip] != ens) {
04422
04423         /* Write results... */
04424         if (n > 0) {
04425
04426             /* Get mean position... */
04427             xm[0] = xm[1] = xm[2] = 0;
04428             for (i = 0; i < n; i++) {
04429                 xm[0] += x[i][0] / (double) n;
04430                 xm[1] += x[i][1] / (double) n;
04431                 xm[2] += x[i][2] / (double) n;
04432             }
04433             cart2geo(xm, &dummy, &lon, &lat);
04434             fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
04435                     lat);
04436
04437             /* Get quantity statistics... */
04438             for (iq = 0; iq < ctl->nq; iq++) {
04439                 fprintf(out, " ");
04440                 fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
04441             }
04442             for (iq = 0; iq < ctl->nq; iq++) {
04443                 fprintf(out, " ");
04444                 fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
04445             }
04446             fprintf(out, " %lu\n", n);
04447         }
04448
04449         /* Init new ensemble... */
04450         ens = atm->q[ctl->qnt_ens][ip];
04451         n = 0;
04452     }
04453
04454     /* Save data... */
04455     p[n] = atm->p[ip];
04456     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
04457     for (iq = 0; iq < ctl->nq; iq++)
04458         q[iq][n] = atm->q[iq][ip];
04459     if ((++n) >= NENS)
04460         ERRMSG("Too many data points!");
04461 }
04462
04463 /* Write results... */
04464 if (n > 0) {
04465
04466     /* Get mean position... */
04467     xm[0] = xm[1] = xm[2] = 0;
04468     for (i = 0; i < n; i++) {
04469         xm[0] += x[i][0] / (double) n;
04470         xm[1] += x[i][1] / (double) n;
04471         xm[2] += x[i][2] / (double) n;
04472     }
04473     cart2geo(xm, &dummy, &lon, &lat);
04474     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
04475
04476     /* Get quantity statistics... */
04477     for (iq = 0; iq < ctl->nq; iq++) {
04478         fprintf(out, " ");
04479         fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
04480     }
04481     for (iq = 0; iq < ctl->nq; iq++) {
04482         fprintf(out, " ");
04483         fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));

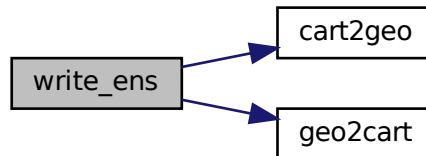
```

```

04484     }
04485     fprintf(out, " %lu\n", n);
04486 }
04487
04488 /* Close file... */
04489 if (t == ctl->t_stop)
04490     fclose(out);
04491 }

```

Here is the call graph for this function:



**5.23.1.48 write\_grid()** 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 4495 of file libtrac.c.

```

04501     {
04502
04503     FILE *in, *out;
04504
04505     char line[LEN];
04506
04507     static double mass[GX][GY][GZ], z[GZ], dz, lon[GX], dlon, lat[GY], dlat,
04508         area[GY], rho_air, press[GZ], temp, cd, vmr, t0, t1, r;
04509
04510     static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec;
04511
04512     /* Set timer... */
04513     SELECT_TIMER("WRITE_GRID", NVTX_WRITE);
04514
04515     /* Check dimensions... */
04516     if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
04517         ERRMSG("Grid dimensions too large!");
04518
04519     /* Set grid box size... */
04520     dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
04521     dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
04522     dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
04523
04524     /* Set vertical coordinates... */
04525     for (iz = 0; iz < ctl->grid_nz; iz++) {
04526         z[iz] = ctl->grid_z0 + dz * (iz + 0.5);
04527         press[iz] = P(z[iz]);
04528     }
04529
04530     /* Set horizontal coordinates... */
04531     for (ix = 0; ix < ctl->grid_nx; ix++)
04532         lon[ix] = ctl->grid_lon0 + dlon * (ix + 0.5);

```

```

04533     for (iy = 0; iy < ctl->grid_ny; iy++) {
04534         lat[iy] = ctl->grid_lat0 + dlat * (iy + 0.5);
04535         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
04536             * cos(lat[iy] * M_PI / 180.);
04537     }
04538
04539     /* Set time interval for output... */
04540     t0 = t - 0.5 * ctl->dt_mod;
04541     t1 = t + 0.5 * ctl->dt_mod;
04542
04543     /* Initialize grid... */
04544     #pragma omp parallel for default(shared) private(ix,iy,iz)
04545     for (ix = 0; ix < ctl->grid_nx; ix++)
04546         for (iy = 0; iy < ctl->grid_ny; iy++)
04547             for (iz = 0; iz < ctl->grid_nz; iz++) {
04548                 mass[ix][iy][iz] = 0;
04549                 np[ix][iy][iz] = 0;
04550             }
04551
04552     /* Average data... */
04553     for (ip = 0; ip < atm->np; ip++)
04554         if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
04555
04556             /* Get index... */
04557             ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
04558             iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
04559             iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
04560
04561             /* Check indices... */
04562             if (ix < 0 || ix >= ctl->grid_nx ||
04563                 iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
04564                 continue;
04565
04566             /* Add mass... */
04567             if (ctl->qnt_m >= 0)
04568                 mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04569             np[ix][iy][iz]++;
04570         }
04571
04572     /* Check if gnuplot output is requested... */
04573     if (ctl->grid_gpfile[0] != '-') {
04574
04575         /* Write info... */
04576         printf("Plot grid data: %s.png\n", filename);
04577
04578         /* Create gnuplot pipe... */
04579         if (!(out = popen("gnuplot", "w")))
04580             ERRMSG("Cannot create pipe to gnuplot!");
04581
04582         /* Set plot filename... */
04583         fprintf(out, "set out \"%s.png\"\\n", filename);
04584
04585         /* Set time string... */
04586         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
04587         fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\\n",
04588             year, mon, day, hour, min);
04589
04590         /* Dump gnuplot file to pipe... */
04591         if (!(in = fopen(ctl->grid_gpfile, "r")))
04592             ERRMSG("Cannot open file!");
04593         while (fgets(line, LEN, in))
04594             fprintf(out, "%s", line);
04595         fclose(in);
04596     }
04597
04598     else {
04599
04600         /* Write info... */
04601         printf("Write grid data: %s\\n", filename);
04602
04603         /* Create file... */
04604         if (!(out = fopen(filename, "w")))
04605             ERRMSG("Cannot create file!");
04606     }
04607
04608     /* Write header... */
04609     fprintf(out,
04610         "# $1 = time [s]\\n"
04611         "# $2 = altitude [km]\\n"
04612         "# $3 = longitude [deg]\\n"
04613         "# $4 = latitude [deg]\\n"
04614         "# $5 = surface area [km^2]\\n"
04615         "# $6 = layer width [km]\\n"
04616         "# $7 = number of particles [l]\\n"
04617         "# $8 = column density [kg/m^2]\\n"
04618         "# $9 = volume mixing ratio [ppv]\\n\\n");
04619

```

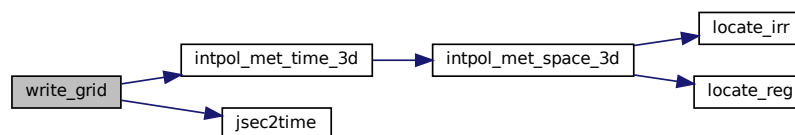


```

04620  /* Write data... */
04621  for (ix = 0; ix < ctl->grid_nx; ix++) {
04622      if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
04623          fprintf(out, "\n");
04624      for (iy = 0; iy < ctl->grid_ny; iy++) {
04625          if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
04626              fprintf(out, "\n");
04627          for (iz = 0; iz < ctl->grid_nz; iz++)
04628              if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
04629
04630                  /* Calculate column density... */
04631                  cd = mass[ix][iy][iz] / (1e6 * area[iy]);
04632
04633                  /* Calculate volume mixing ratio... */
04634                  vmr = 0;
04635                  if (ctl->molmass > 0) {
04636                      if (mass[ix][iy][iz] > 0) {
04637
04638                          /* Get temperature... */
04639                          INTPOL_INIT;
04640                          intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
04641                                              lon[ix], lat[iy], &temp, ci, cw, 1);
04642
04643                          /* Calculate density of air... */
04644                          rho_air = 100. * press[iz] / (RA * temp);
04645
04646                          /* Calculate volume mixing ratio... */
04647                          vmr = MA / ctl->molmass * mass[ix][iy][iz]
04648                              / (rho_air * 1e6 * area[iy] * 1e3 * dz);
04649                      }
04650                  } else
04651                      vmr = GSL_NAN;
04652
04653                  /* Write output... */
04654                  fprintf(out, "%.2f %g %g %g %g %d %g %g\n", t, z[iz],
04655                          lon[ix], lat[iy], area[iy], dz, np[ix][iy][iz], cd, vmr);
04656              }
04657          }
04658      }
04659
04660      /* Close file... */
04661      fclose(out);
04662  }

```

Here is the call graph for this function:



**5.23.1.49 write\_prof()** 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 4666 of file libtrac.c.

```

04672     {
04673

```

```

04674 static FILE *in, *out;
04675
04676 static char line[LEN];
04677
04678 static double mass[GX][GY][GZ], obsmean[GX][GY], rt, rt_old = -1e99,
04679 rz, rlon, rlat, robs, t0, t1, area[GY], dz, dlon, dlat, lon[GX], lat[GY],
04680 z[GZ], press[GZ], temp, rho_air, vmr, h2o, o3;
04681
04682 static int obscount[GX][GY], ip, ix, iy, iz, okay;
04683
04684 /* Set timer... */
04685 SELECT_TIMER("WRITE_PROF", NVTX_WRITE);
04686
04687 /* Init... */
04688 if (t == ctl->t_start) {
04689
04690     /* Check quantity index for mass... */
04691     if (ctl->qnt_m < 0)
04692         ERRMSG("Need quantity mass!");
04693
04694     /* Check dimensions... */
04695     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
04696         ERRMSG("Grid dimensions too large!");
04697
04698     /* Check molar mass... */
04699     if (ctl->molmass <= 0)
04700         ERRMSG("Specify molar mass!");
04701
04702     /* Open observation data file... */
04703     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
04704     if (!(in = fopen(ctl->prof_obsfile, "r")))
04705         ERRMSG("Cannot open file!");
04706
04707     /* Create new output file... */
04708     printf("Write profile data: %s\n", filename);
04709     if (!(out = fopen(filename, "w")))
04710         ERRMSG("Cannot create file!");
04711
04712     /* Write header... */
04713     fprintf(out,
04714         "# $1 = time [s]\n"
04715         "# $2 = altitude [km]\n"
04716         "# $3 = longitude [deg]\n"
04717         "# $4 = latitude [deg]\n"
04718         "# $5 = pressure [hPa]\n"
04719         "# $6 = temperature [K]\n"
04720         "# $7 = volume mixing ratio [ppv]\n"
04721         "# $8 = H2O volume mixing ratio [ppv]\n"
04722         "# $9 = O3 volume mixing ratio [ppv]\n"
04723         "# $10 = observed BT index [K]\n");
04724
04725     /* Set grid box size... */
04726     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
04727     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
04728     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
04729
04730     /* Set vertical coordinates... */
04731     for (iz = 0; iz < ctl->prof_nz; iz++) {
04732         z[iz] = ctl->prof_z0 + dz * (iz + 0.5);
04733         press[iz] = P(z[iz]);
04734     }
04735
04736     /* Set horizontal coordinates... */
04737     for (ix = 0; ix < ctl->prof_nx; ix++)
04738         lon[ix] = ctl->prof_lon0 + dlon * (ix + 0.5);
04739     for (iy = 0; iy < ctl->prof_ny; iy++) {
04740         lat[iy] = ctl->prof_lat0 + dlat * (iy + 0.5);
04741         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
04742             * cos(lat[iy] * M_PI / 180.);
04743     }
04744 }
04745
04746 /* Set time interval... */
04747 t0 = t - 0.5 * ctl->dt_mod;
04748 t1 = t + 0.5 * ctl->dt_mod;
04749
04750 /* Initialize... */
04751 #pragma omp parallel for default(shared) private(ix,iy,iz)
04752 for (ix = 0; ix < ctl->prof_nx; ix++)
04753     for (iy = 0; iy < ctl->prof_ny; iy++) {
04754         obsmean[ix][iy] = 0;
04755         obscount[ix][iy] = 0;
04756         for (iz = 0; iz < ctl->prof_nz; iz++)
04757             mass[ix][iy][iz] = 0;
04758     }
04759
04760 /* Read observation data... */

```

```

04761 while (fgets(line, LEN, in)) {
04762
04763     /* Read data... */
04764     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &robs) !=
04765         5)
04766         continue;
04767
04768     /* Check time... */
04769     if (rt < t0)
04770         continue;
04771     if (rt > t1)
04772         break;
04773     if (rt < rt_old)
04774         ERRMSG("Time must be ascending!");
04775     rt_old = rt;
04776
04777     /* Check observation data... */
04778     if (!isfinite(robs))
04779         continue;
04780
04781     /* Calculate indices... */
04782     ix = (int) ((rln - ctl->prof_lon0) / dlon);
04783     iy = (int) ((rln - ctl->prof_lat0) / dlat);
04784
04785     /* Check indices... */
04786     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
04787         continue;
04788
04789     /* Get mean observation index... */
04790     obsmean[ix][iy] += robs;
04791     obscount[ix][iy]++;
04792 }
04793
04794 /* Analyze model data... */
04795 for (ip = 0; ip < atm->np; ip++) {
04796
04797     /* Check time... */
04798     if (atm->time[ip] < t0 || atm->time[ip] > t1)
04799         continue;
04800
04801     /* Get indices... */
04802     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
04803     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
04804     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
04805
04806     /* Check indices... */
04807     if (ix < 0 || ix >= ctl->prof_nx ||
04808         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
04809         continue;
04810
04811     /* Get total mass in grid cell... */
04812     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04813 }
04814
04815 /* Extract profiles... */
04816 for (ix = 0; ix < ctl->prof_nx; ix++)
04817     for (iy = 0; iy < ctl->prof_ny; iy++)
04818         if (obscount[ix][iy] > 0) {
04819
04820             /* Check profile... */
04821             okay = 0;
04822             for (iz = 0; iz < ctl->prof_nz; iz++)
04823                 if (mass[ix][iy][iz] > 0) {
04824                     okay = 1;
04825                     break;
04826                 }
04827             if (!okay)
04828                 continue;
04829
04830             /* Write output... */
04831             fprintf(out, "\n");
04832
04833             /* Loop over altitudes... */
04834             for (iz = 0; iz < ctl->prof_nz; iz++) {
04835
04836                 /* Get pressure and temperature... */
04837                 INTPOL_INIT;
04838                 intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
04839                                     lon[ix], lat[iy], &temp, ci, cw, 1);
04840                 intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, t, press[iz],
04841                                     lon[ix], lat[iy], &h2o, ci, cw, 0);
04842                 intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press[iz],
04843                                     lon[ix], lat[iy], &o3, ci, cw, 0);
04844
04845                 /* Calculate volume mixing ratio... */
04846                 rho_air = 100. * press[iz] / (RA * temp);
04847                 vmr = MA / ctl->molmass * mass[ix][iy][iz]

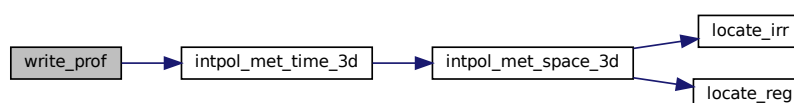
```

```

04848         / (rho_air * area[iy] * dz * 1e9);
04849
04850     /* Write output... */
04851     fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
04852            t, z[iz], lon[ix], lat[iy], press[iz], temp, vmr, h2o, o3,
04853            obsmean[ix][iy] / obscount[ix][iy]);
04854     }
04855 }
04856
04857 /* Close files... */
04858 if (t == ctl->t_stop) {
04859     fclose(in);
04860     fclose(out);
04861 }
04862 }

```

Here is the call graph for this function:



**5.23.1.50 write\_sample()** void write\_sample (

```

    const char * filename,
    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double t )

```

Write sample data.

Definition at line 4866 of file libtrac.c.

```

04872     {
04873
04874     static FILE *in, *out;
04875
04876     static char line[LEN];
04877
04878     static double area, dlat, rmax2, t0, t1, rt, rt_old, rz, rlon, rlat, robs;
04879
04880     /* Set timer... */
04881     SELECT_TIMER("WRITE_SAMPLE", NVTX_WRITE);
04882
04883     /* Init... */
04884     if (t == ctl->t_start) {
04885
04886         /* Open observation data file... */
04887         printf("Read sample observation data: %s\n", ctl->sample_obsfile);
04888         if (!(in = fopen(ctl->sample_obsfile, "r")))
04889             ERRMSG("Cannot open file!");
04890
04891         /* Create new file... */
04892         printf("Write sample data: %s\n", filename);
04893         if (!(out = fopen(filename, "w")))
04894             ERRMSG("Cannot create file!");
04895
04896         /* Write header... */
04897         fprintf(out,
04898             "# $1 = time [s]\n"
04899             "# $2 = altitude [km]\n"
04900             "# $3 = longitude [deg]\n"
04901             "# $4 = latitude [deg]\n"
04902             "# $5 = surface area [km^2]\n"

```

```

04903         "# $6 = layer width [km]\n"
04904         "# $7 = number of particles [l]\n"
04905         "# $8 = column density [kg/m^2]\n"
04906         "# $9 = volume mixing ratio [ppv]\n"
04907         "# $10 = observed BT index [K]\n\n");
04908
04909     /* Set latitude range, squared radius, and area... */
04910     dlat = DY2DEG(ctl->sample_dx);
04911     rmax2 = SQR(ctl->sample_dx);
04912     area = M_PI * rmax2;
04913 }
04914
04915 /* Set time interval for output... */
04916 t0 = t - 0.5 * ctl->dt_mod;
04917 t1 = t + 0.5 * ctl->dt_mod;
04918
04919 /* Read observation data... */
04920 while (fgets(line, LEN, in)) {
04921
04922     /* Read data... */
04923     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &rln) !=
04924         5)
04925         continue;
04926
04927     /* Check time... */
04928     if (rt < t0)
04929         continue;
04930     if (rt < rt_old)
04931         ERRMSG("Time must be ascending!");
04932     rt_old = rt;
04933
04934     /* Calculate Cartesian coordinates... */
04935     double x0[3];
04936     geo2cart(0, rln, rln, x0);
04937
04938     /* Set pressure range... */
04939     double rp = P(rz);
04940     double ptop = P(rz + ctl->sample_dz);
04941     double pbot = P(rz - ctl->sample_dz);
04942
04943     /* Init... */
04944     double mass = 0;
04945     int np = 0;
04946
04947     /* Loop over air parcels... */
04948 #pragma omp parallel for default(shared) reduction(+:mass,np)
04949     for (int ip = 0; ip < atm->np; ip++) {
04950
04951         /* Check time... */
04952         if (atm->time[ip] < t0 || atm->time[ip] > t1)
04953             continue;
04954
04955         /* Check latitude... */
04956         if (fabs(rln - atm->lat[ip]) > dlat)
04957             continue;
04958
04959         /* Check horizontal distance... */
04960         double x1[3];
04961         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
04962         if (DIST2(x0, x1) > rmax2)
04963             continue;
04964
04965         /* Check pressure... */
04966         if (ctl->sample_dz > 0)
04967             if (atm->p[ip] > pbot || atm->p[ip] < ptop)
04968                 continue;
04969
04970         /* Add mass... */
04971         if (ctl->qnt_m >= 0)
04972             mass += atm->q[ctl->qnt_m][ip];
04973         np++;
04974     }
04975
04976     /* Calculate column density... */
04977     double cd = mass / (1e6 * area);
04978
04979     /* Calculate volume mixing ratio... */
04980     double vmr = 0;
04981     if (ctl->molmass > 0 && ctl->sample_dz > 0) {
04982         if (mass > 0) {
04983
04984             /* Get temperature... */
04985             double temp;
04986             INTPOL_INIT;
04987             intpol_met_time_3d(met0, met0->t, met1, met1->t, rt, rp,
04988                             rln, rln, &temp, ci, cw, 1);
04989

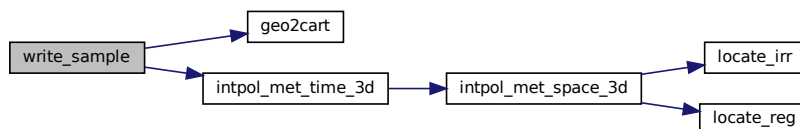
```

```

04990      /* Calculate density of air... */
04991      double rho_air = 100. * rp / (RA * temp);
04992
04993      /* Calculate volume mixing ratio... */
04994      vmr = MA / ctl->molmass * mass
04995            / (rho_air * 1e6 * area * 1e3 * ctl->sample_dz);
04996  }
04997  } else
04998  {
04999      vmr = GSL_NAN;
05000
05001      /* Write output... */
05002      fprintf(out, "%.2f %g %g %g %g %d %g %g %g\n", rt, rz, rlon, rlat,
05003            area, ctl->sample_dz, np, cd, vmr, robs);
05004
05005      /* Check time... */
05006      if (rt >= t1)
05007          break;
05008  }
05009
05010      /* Close files... */
05011      if (t == ctl->t_stop) {
05012          fclose(in);
05013          fclose(out);
05014      }

```

Here is the call graph for this function:



**5.23.1.51 write\_station()** void write\_station (

```

    const char * filename,
    ctl_t * ctl,
    atm_t * atm,
    double t )

```

Write station data.

Definition at line 5018 of file libtrac.c.

```

05022      {
05023
05024      static FILE *out;
05025
05026      static double rmax2, t0, t1, x0[3], x1[3];
05027
05028      /* Set timer... */
05029      SELECT_TIMER("WRITE_STATION", NVTX_WRITE);
05030
05031      /* Init... */
05032      if (t == ctl->t_start) {
05033
05034          /* Write info... */
05035          printf("Write station data: %s\n", filename);
05036
05037          /* Create new file... */
05038          if (!(out = fopen(filename, "w")))
05039              ERRMSG("Cannot create file!");
05040
05041          /* Write header... */
05042          fprintf(out,
05043                "# $1 = time [s]\n"
05044                "# $2 = altitude [km]\n"

```

```

05045         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
05046     for (int iq = 0; iq < ctl->nq; iq++)
05047         fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
05048             ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05049     fprintf(out, "\n");
05050
05051     /* Set geolocation and search radius... */
05052     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
05053     rmax2 = SQR(ctl->stat_r);
05054 }
05055
05056 /* Set time interval for output... */
05057 t0 = t - 0.5 * ctl->dt_mod;
05058 t1 = t + 0.5 * ctl->dt_mod;
05059
05060 /* Loop over air parcels... */
05061 for (int ip = 0; ip < atm->np; ip++) {
05062
05063     /* Check time... */
05064     if (atm->time[ip] < t0 || atm->time[ip] > t1)
05065         continue;
05066
05067     /* Check station flag... */
05068     if (ctl->qnt_stat >= 0)
05069         if (atm->q[ctl->qnt_stat][ip])
05070             continue;
05071
05072     /* Get Cartesian coordinates... */
05073     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
05074
05075     /* Check horizontal distance... */
05076     if (DIST2(x0, x1) > rmax2)
05077         continue;
05078
05079     /* Set station flag... */
05080     if (ctl->qnt_stat >= 0)
05081         atm->q[ctl->qnt_stat][ip] = 1;
05082
05083     /* Write data... */
05084     fprintf(out, "%.2f %g %g %g",
05085         atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
05086     for (int iq = 0; iq < ctl->nq; iq++) {
05087         fprintf(out, " ");
05088         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
05089     }
05090     fprintf(out, "\n");
05091 }
05092
05093 /* Close file... */
05094 if (t == ctl->t_stop)
05095     fclose(out);
05096 }

```

Here is the call graph for this function:



## 5.24 libtrac.h

```

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
00015 along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00035 /* -----
00036 Includes...
00037 ----- */
00038
00039 #include <ctype.h>
00040 #include <gsl/gsl_fft_complex.h>
00041 #include <gsl/gsl_math.h>
00042 #include <gsl/gsl_randist.h>
00043 #include <gsl/gsl_rng.h>
00044 #include <gsl/gsl_sort.h>
00045 #include <gsl/gsl_spline.h>
00046 #include <gsl/gsl_statistics.h>
00047 #include <math.h>
00048 #include <netcdf.h>
00049 #include <omp.h>
00050 #include <stdio.h>
00051 #include <stdlib.h>
00052 #include <string.h>
00053 #include <time.h>
00054 #include <sys/time.h>
00055
00056 #ifdef MPI
00057 #include "mpi.h"
00058 #endif
00059
00060 #ifdef _OPENACC
00061 #include "openacc.h"
00062 #include "curand.h"
00063 #endif
00064
00065 /* -----
00066 Constants...
00067 ----- */
00068
00070 #define CPD 1003.5
00071
00073 #define EPS (MH20 / MA)
00074
00076 #define G0 9.80665
00077
00079 #define H0 7.0
00080
00082 #define LV 2501000.
00083
00085 #define KB 1.3806504e-23
00086
00088 #define MA 28.9644
00089
00091 #define MH20 18.01528
00092
00094 #define MO3 48.00
00095
00097 #define P0 1013.25
00098
00100 #define RA (1e3 * RI / MA)
00101
00103 #define RE 6367.421
00104
00106 #define RI 8.3144598
00107
00109 #define T0 273.15
00110
00111 /* -----
00112 Dimensions...
00113 ----- */
00114
00116 #define LEN 5000
00117
00119 #define NP 10000000
00120
00122 #define NQ 15
00123
00125 #define EP 140
00126
00128 #define EX 1201
00129
00131 #define EY 601
00132
00134 #define GX 720
```



```

00135
00137 #define GY 360
00138
00140 #define GZ 100
00141
00143 #define NENS 2000
00144
00146 #define NTHREADS 512
00147
00148 /* -----
00149     Macros...
00150 ----- */
00151
00153 #define ALLOC(ptr, type, n) \
00154     if((ptr=calloc((size_t)(n), sizeof(type)))==NULL) \
00155         ERRMSG("Out of memory!");
00156
00158 #define ATM_SET(qnt, val) \
00159     if (ctl->qnt >= 0) \
00160         atm->q[ctl->qnt][ip] = val;
00161
00163 #define DEG2DX(dlon, lat) \
00164     ((dlon) * M_PI * RE / 180. * cos((lat) / 180. * M_PI))
00165
00167 #define DEG2DY(dlat) \
00168     ((dlat) * M_PI * RE / 180.)
00169
00171 #define DP2DZ(dp, p) \
00172     (- (dp) * H0 / (p))
00173
00175 #define DX2DEG(dx, lat) \
00176     (((lat) < -89.999 || (lat) > 89.999) ? 0 \
00177      : (dx) * 180. / (M_PI * RE * cos((lat) / 180. * M_PI)))
00178
00180 #define DY2DEG(dy) \
00181     ((dy) * 180. / (M_PI * RE))
00182
00184 #define DZ2DP(dz, p) \
00185     (- (dz) * (p) / H0)
00186
00188 #define DIST(a, b) sqrt(DIST2(a, b))
00189
00191 #define DIST2(a, b) \
00192     ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))
00193
00195 #define DOTP(a, b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
00196
00198 #define ERRMSG(msg) { \
00199     printf("\nError (%s, %s, %d): %s\n\n", \
00200         __FILE__, __func__, __LINE__, msg); \
00201     exit(EXIT_FAILURE); \
00202 }
00203
00205 #define FMOD(x, y) \
00206     ((x) - (int) ((x) / (y)) * (y))
00207
00209 #define FREAD(ptr, type, size, out) { \
00210     if(fread(ptr, sizeof(type), size, out)!=size) \
00211         ERRMSG("Error while reading!"); \
00212 }
00213
00215 #define FWRITE(ptr, type, size, out) { \
00216     if(fwrite(ptr, sizeof(type), size, out)!=size) \
00217         ERRMSG("Error while writing!"); \
00218 }
00219
00221 #define INTPOL_INIT \
00222     double cw[3] = {0.0, 0.0, 0.0}; int ci[3] = {0, 0, 0};
00223
00225 #define INTPOL_2D(var, init) \
00226     intpol_met_time_2d(met0, met0->var, met1, met1->var, \
00227         atm->time[ip], atm->lon[ip], atm->lat[ip], \
00228         &var, ci, cw, init);
00229
00231 #define INTPOL_3D(var, init) \
00232     intpol_met_time_3d(met0, met0->var, met1, met1->var, \
00233         atm->time[ip], atm->p[ip], \
00234         atm->lon[ip], atm->lat[ip], \
00235         &var, ci, cw, init);
00236
00238 #define INTPOL_SPACE_ALL(p, lon, lat) { \
00239     intpol_met_space_3d(met, met->z, p, lon, lat, &z, ci, cw, 1); \
00240     intpol_met_space_3d(met, met->t, p, lon, lat, &t, ci, cw, 0); \
00241     intpol_met_space_3d(met, met->u, p, lon, lat, &u, ci, cw, 0); \
00242     intpol_met_space_3d(met, met->v, p, lon, lat, &v, ci, cw, 0); \
00243     intpol_met_space_3d(met, met->w, p, lon, lat, &w, ci, cw, 0); \
00244     intpol_met_space_3d(met, met->pv, p, lon, lat, &pv, ci, cw, 0); \

```

```

00245 intpol_met_space_3d(met, met->h2o, p, lon, lat, &h2o, ci, cw, 0); \
00246 intpol_met_space_3d(met, met->o3, p, lon, lat, &o3, ci, cw, 0); \
00247 intpol_met_space_3d(met, met->lwc, p, lon, lat, &lwc, ci, cw, 0); \
00248 intpol_met_space_3d(met, met->iwc, p, lon, lat, &iwc, ci, cw, 0); \
00249 intpol_met_space_2d(met, met->ps, lon, lat, &ps, ci, cw, 0); \
00250 intpol_met_space_2d(met, met->ts, lon, lat, &ts, ci, cw, 0); \
00251 intpol_met_space_2d(met, met->zs, lon, lat, &zs, ci, cw, 0); \
00252 intpol_met_space_2d(met, met->us, lon, lat, &us, ci, cw, 0); \
00253 intpol_met_space_2d(met, met->vs, lon, lat, &vs, ci, cw, 0); \
00254 intpol_met_space_2d(met, met->pt, lon, lat, &pt, ci, cw, 0); \
00255 intpol_met_space_2d(met, met->tt, lon, lat, &tt, ci, cw, 0); \
00256 intpol_met_space_2d(met, met->zt, lon, lat, &zt, ci, cw, 0); \
00257 intpol_met_space_2d(met, met->h2ot, lon, lat, &h2ot, ci, cw, 0); \
00258 intpol_met_space_2d(met, met->pc, lon, lat, &pc, ci, cw, 0); \
00259 intpol_met_space_2d(met, met->c1, lon, lat, &c1, ci, cw, 0); \
00260 intpol_met_space_2d(met, met->plcl, lon, lat, &plcl, ci, cw, 0); \
00261 intpol_met_space_2d(met, met->plfc, lon, lat, &plfc, ci, cw, 0); \
00262 intpol_met_space_2d(met, met->pel, lon, lat, &pel, ci, cw, 0); \
00263 intpol_met_space_2d(met, met->cape, lon, lat, &cape, ci, cw, 0); \
00264 }
00265
00267 #define INTPOL_TIME_ALL(time, p, lon, lat) { \
00268 intpol_met_time_3d(met0, met0->z, met1, met1->z, time, p, lon, lat, &z, ci, cw, 1); \
00269 intpol_met_time_3d(met0, met0->t, met1, met1->t, time, p, lon, lat, &t, ci, cw, 0); \
00270 intpol_met_time_3d(met0, met0->u, met1, met1->u, time, p, lon, lat, &u, ci, cw, 0); \
00271 intpol_met_time_3d(met0, met0->v, met1, met1->v, time, p, lon, lat, &v, ci, cw, 0); \
00272 intpol_met_time_3d(met0, met0->w, met1, met1->w, time, p, lon, lat, &w, ci, cw, 0); \
00273 intpol_met_time_3d(met0, met0->pv, met1, met1->pv, time, p, lon, lat, &pv, ci, cw, 0); \
00274 intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, time, p, lon, lat, &h2o, ci, cw, 0); \
00275 intpol_met_time_3d(met0, met0->o3, met1, met1->o3, time, p, lon, lat, &o3, ci, cw, 0); \
00276 intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, time, p, lon, lat, &lwc, ci, cw, 0); \
00277 intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, time, p, lon, lat, &iwc, ci, cw, 0); \
00278 intpol_met_time_2d(met0, met0->ps, met1, met1->ps, time, lon, lat, &ps, ci, cw, 0); \
00279 intpol_met_time_2d(met0, met0->ts, met1, met1->ts, time, lon, lat, &ts, ci, cw, 0); \
00280 intpol_met_time_2d(met0, met0->zs, met1, met1->zs, time, lon, lat, &zs, ci, cw, 0); \
00281 intpol_met_time_2d(met0, met0->us, met1, met1->us, time, lon, lat, &us, ci, cw, 0); \
00282 intpol_met_time_2d(met0, met0->vs, met1, met1->vs, time, lon, lat, &vs, ci, cw, 0); \
00283 intpol_met_time_2d(met0, met0->pt, met1, met1->pt, time, lon, lat, &pt, ci, cw, 0); \
00284 intpol_met_time_2d(met0, met0->tt, met1, met1->tt, time, lon, lat, &tt, ci, cw, 0); \
00285 intpol_met_time_2d(met0, met0->zt, met1, met1->zt, time, lon, lat, &zt, ci, cw, 0); \
00286 intpol_met_time_2d(met0, met0->h2ot, met1, met1->h2ot, time, lon, lat, &h2ot, ci, cw, 0); \
00287 intpol_met_time_2d(met0, met0->pc, met1, met1->pc, time, lon, lat, &pc, ci, cw, 0); \
00288 intpol_met_time_2d(met0, met0->c1, met1, met1->c1, time, lon, lat, &c1, ci, cw, 0); \
00289 intpol_met_time_2d(met0, met0->plcl, met1, met1->plcl, time, lon, lat, &plcl, ci, cw, 0); \
00290 intpol_met_time_2d(met0, met0->plfc, met1, met1->plfc, time, lon, lat, &plfc, ci, cw, 0); \
00291 intpol_met_time_2d(met0, met0->pel, met1, met1->pel, time, lon, lat, &pel, ci, cw, 0); \
00292 intpol_met_time_2d(met0, met0->cape, met1, met1->cape, time, lon, lat, &cape, ci, cw, 0); \
00293 }
00294
00296 #define LAPSE(pl, t1, p2, t2) \
00297 (le3 * G0 / RA * ((t2) - (t1)) / ((t2) + (t1)) * ((p2) + (p1)) / ((p2) - (p1)))
00298
00300 #define LIN(x0, y0, x1, y1, x) \
00301 ((y0) + ((y1) - (y0)) / ((x1) - (x0)) * ((x) - (x0)))
00302
00304 #define NC(cmd) { \
00305 if ((cmd) != NC_NOERR) \
00306 ERRMSG(nc_strerror(cmd)); \
00307 }
00308
00310 #define NN(x0, y0, x1, y1, x) \
00311 (fabs((x) - (x0)) <= fabs((x) - (x1)) ? (y0) : (y1))
00312
00314 #define NORM(a) sqrt(DOTP(a, a))
00315
00317 #define PRINT(format, var) \
00318 printf("Print (%s, %s, l%d): %s= "format"\n", \
00319 FILE, func, LINE, #var, var);
00320
00322 #define P(z) (P0 * exp(-(z) / H0))
00323
00325 #define PSAT(t) \
00326 (6.112 * exp(17.62 * ((t) - T0) / (243.12 + (t) - T0)))
00327
00329 #define PSICE(t) \
00330 (0.01 * pow(10., -2663.5 / (t) + 12.537))
00331
00333 #define PW(p, h2o) \
00334 ((p) * (h2o) / (1. + (1. - EPS) * (h2o)))
00335
00337 #define RH(p, t, h2o) \
00338 (PW(p, h2o) / PSAT(t) * 100.)
00339
00341 #define RHICE(p, t, h2o) \
00342 (PW(p, h2o) / PSICE(t) * 100.)
00343
00345 #define SH(h2o) (EPS * (h2o))

```

```

00346
00348 #define SQR(x) ((x)*(x))
00349
00351 #define TDEW(p, h2o) \
00352     (T0 + 243.12 * log(PW((p), (h2o)) / 6.112) \
00353     / (17.62 - log(PW((p), (h2o)) / 6.112)))
00354
00356 #define TICE(p, h2o) \
00357     (-2663.5 / (log10(100. * PW((p), (h2o))) - 12.537))
00358
00360 #define THETA(p, t) ((t) * pow(1000. / (p), 0.286))
00361
00363 #define TOK(line, tok, format, var) { \
00364     if((tok)=strtok((line), " \t")) { \
00365         if(sscanf(tok, format, &(var))!=1) continue; \
00366     } else ERRMSG("Error while reading!"); \
00367 }
00368
00370 #define TVIRT(t, h2o) \
00371     ((t) * (1. + (1. - EPS) * (h2o)))
00372
00374 #define WARN(msg) { \
00375     printf("\nWarning (%s, %s, %d): %s\n\n", \
00376         __FILE__, __func__, __LINE__, msg); \
00377 }
00378
00380 #define Z(p) (H0 * log(P0 / (p)))
00381
00383 #define ZDIFF(lnp0, t0, h2o0, lnp1, t1, h2o1) \
00384     (RI / MA / G0 * 0.5 * (TVIRT((t0), (h2o0)) + TVIRT((t1), (h2o1))) * ((lnp0) - (lnp1)))
00385
00386 /* -----
00387     Timers...
00388     ----- */
00389
00391 #define NTIMER 100
00392
00394 #define PRINT_TIMERS \
00395     timer("END", 1);
00396
00398 #define SELECT_TIMER(id, color) \
00399     {NVTX_POP; NVTX_PUSH(id, color); timer(id, 0);}
00400
00402 #define START_TIMERS \
00403     NVTX_PUSH("START", NVTX_CPU);
00404
00406 #define STOP_TIMERS \
00407     NVTX_POP;
00408
00409 /* -----
00410     NVIDIA Tools Extension (NVTX)...
00411     ----- */
00412
00413 #ifdef NVTX
00414 #include "nvToolsExt.h"
00415
00417 #define NVTX_CPU 0xFFADD8E6
00418
00420 #define NVTX_GPU 0xFF00008B
00421
00423 #define NVTX_H2D 0xFFFFFFFF00
00424
00426 #define NVTX_D2H 0xFFFF8800
00427
00429 #define NVTX_READ 0xFFFFCCCB
00430
00432 #define NVTX_WRITE 0xFF8B0000
00433
00435 #define NVTX_MISC 0xFF808080
00436
00438 #define NVTX_PUSH(range_title, range_color) { \
00439     nvtxEventAttributes_t eventAttrib = {0}; \
00440     eventAttrib.version = NVTX_VERSION; \
00441     eventAttrib.size = NVTX_EVENT_ATTRIB_STRUCT_SIZE; \
00442     eventAttrib.messageType = NVTX_MESSAGE_TYPE_ASCII; \
00443     eventAttrib.colorType = NVTX_COLOR_ARGB; \
00444     eventAttrib.color = range_color; \
00445     eventAttrib.message.ascii = range_title; \
00446     nvtxRangePushEx(&eventAttrib); \
00447 }
00448
00450 #define NVTX_POP { \
00451     nvtxRangePop(); \
00452 }
00453 #else
00454
00455 /* Empty definitions of NVTX_PUSH and NVTX_POP... */

```

```
00456 #define NVTX_PUSH(range_title, range_color) {}
00457 #define NVTX_POP {}
00458 #endif
00459
00460 /* -----
00461    Structs...
00462    ----- */
00463
00464 typedef struct {
00465     int nq;
00466
00467     char qnt_name[NQ][LEN];
00472
00474     char qnt_unit[NQ][LEN];
00475
00477     char qnt_format[NQ][LEN];
00478
00480     int qnt_ens;
00481
00483     int qnt_m;
00484
00486     int qnt_rho;
00487
00489     int qnt_r;
00490
00492     int qnt_ps;
00493
00495     int qnt_ts;
00496
00498     int qnt_zs;
00499
00501     int qnt_us;
00502
00504     int qnt_vs;
00505
00507     int qnt_pt;
00508
00510     int qnt_tt;
00511
00513     int qnt_zt;
00514
00516     int qnt_h2ot;
00517
00519     int qnt_z;
00520
00522     int qnt_p;
00523
00525     int qnt_t;
00526
00528     int qnt_u;
00529
00531     int qnt_v;
00532
00534     int qnt_w;
00535
00537     int qnt_h2o;
00538
00540     int qnt_o3;
00541
00543     int qnt_lwc;
00544
00546     int qnt_iwc;
00547
00549     int qnt_pc;
00550
00552     int qnt_cl;
00553
00555     int qnt_plcl;
00556
00558     int qnt_plfc;
00559
00561     int qnt_pel;
00562
00564     int qnt_cape;
00565
00567     int qnt_hno3;
00568
00570     int qnt_oh;
00571
00573     int qnt_psat;
00574
00576     int qnt_psice;
00577
00579     int qnt_pw;
00580
00582     int qnt_sh;
```

```
00583
00585     int qnt_rh;
00586
00588     int qnt_rhice;
00589
00591     int qnt_theta;
00592
00594     int qnt_tvirt;
00595
00597     int qnt_lapse;
00598
00600     int qnt_vh;
00601
00603     int qnt_vz;
00604
00606     int qnt_pv;
00607
00609     int qnt_tdew;
00610
00612     int qnt_tice;
00613
00615     int qnt_tsts;
00616
00618     int qnt_tnat;
00619
00621     int qnt_stat;
00622
00624     int direction;
00625
00627     double t_start;
00628
00630     double t_stop;
00631
00633     double dt_mod;
00634
00636     char metbase[LEN];
00637
00639     double dt_met;
00640
00642     int met_dx;
00643
00645     int met_dy;
00646
00648     int met_dp;
00649
00651     int met_sx;
00652
00654     int met_sy;
00655
00657     int met_sp;
00658
00660     double met_detrend;
00661
00663     int met_np;
00664
00666     double met_p[EP];
00667
00669     int met_geopot_sx;
00670
00672     int met_geopot_sy;
00673
00676     int met_tropo;
00677
00679     double met_dt_out;
00680
00683     int isosurf;
00684
00686     char balloon[LEN];
00687
00689     double turb_dx_trop;
00690
00692     double turb_dx_strat;
00693
00695     double turb_dz_trop;
00696
00698     double turb_dz_strat;
00699
00701     double turb_mesox;
00702
00704     double turb_mesoz;
00705
00707     double conv_cape;
00708
00710     char species[LEN];
00711
00713     double molmass;
00714
```

```
00716 double tdec_trop;
00717
00719 double tdec_strat;
00720
00722 double oh_chem[4];
00723
00725 double dry_depo[1];
00726
00728 double wet_depo[8];
00729
00731 double psc_h2o;
00732
00734 double psc_hno3;
00735
00737 char atm_basename[LEN];
00738
00740 char atm_gpfile[LEN];
00741
00743 double atm_dt_out;
00744
00746 int atm_filter;
00747
00749 int atm_stride;
00750
00752 int atm_type;
00753
00755 char csi_basename[LEN];
00756
00758 double csi_dt_out;
00759
00761 char csi_obsfile[LEN];
00762
00764 double csi_obsmin;
00765
00767 double csi_modmin;
00768
00770 int csi_nz;
00771
00773 double csi_z0;
00774
00776 double csi_z1;
00777
00779 int csi_nx;
00780
00782 double csi_lon0;
00783
00785 double csi_lon1;
00786
00788 int csi_ny;
00789
00791 double csi_lat0;
00792
00794 double csi_lat1;
00795
00797 char grid_basename[LEN];
00798
00800 char grid_gpfile[LEN];
00801
00803 double grid_dt_out;
00804
00806 int grid_sparse;
00807
00809 int grid_nz;
00810
00812 double grid_z0;
00813
00815 double grid_z1;
00816
00818 int grid_nx;
00819
00821 double grid_lon0;
00822
00824 double grid_lon1;
00825
00827 int grid_ny;
00828
00830 double grid_lat0;
00831
00833 double grid_lat1;
00834
00836 char prof_basename[LEN];
00837
00839 char prof_obsfile[LEN];
00840
00842 int prof_nz;
00843
00845 double prof_z0;
```

```
00846
00848 double prof_z1;
00849
00851 int prof_nx;
00852
00854 double prof_lon0;
00855
00857 double prof_lon1;
00858
00860 int prof_ny;
00861
00863 double prof_lat0;
00864
00866 double prof_lat1;
00867
00869 char ens_basename[LEN];
00870
00872 char sample_basename[LEN];
00873
00875 char sample_obsfile[LEN];
00876
00878 double sample_dx;
00879
00881 double sample_dz;
00882
00884 char stat_basename[LEN];
00885
00887 double stat_lon;
00888
00890 double stat_lat;
00891
00893 double stat_r;
00894
00895 } ctl_t;
00896
00898 typedef struct {
00899
00901 int np;
00902
00904 double time[NP];
00905
00907 double p[NP];
00908
00910 double lon[NP];
00911
00913 double lat[NP];
00914
00916 double q[NQ][NP];
00917
00918 } atm_t;
00919
00921 typedef struct {
00922
00924 float up[NP];
00925
00927 float vp[NP];
00928
00930 float wp[NP];
00931
00933 double iso_var[NP];
00934
00936 double iso_ps[NP];
00937
00939 double iso_ts[NP];
00940
00942 int iso_n;
00943
00945 double tsig[EX][EY][EP];
00946
00948 float usig[EX][EY][EP];
00949
00951 float vsig[EX][EY][EP];
00952
00954 float wsig[EX][EY][EP];
00955
00956 } cache_t;
00957
00959 typedef struct {
00960
00962 double time;
00963
00965 int nx;
00966
00968 int ny;
00969
00971 int np;
00972
```

```

00974 double lon[EX];
00975
00977 double lat[EY];
00978
00980 double p[EP];
00981
00983 float ps[EX][EY];
00984
00986 float ts[EX][EY];
00987
00989 float zs[EX][EY];
00990
00992 float us[EX][EY];
00993
00995 float vs[EX][EY];
00996
00998 float pt[EX][EY];
00999
01001 float tt[EX][EY];
01002
01004 float zt[EX][EY];
01005
01007 float h2ot[EX][EY];
01008
01010 float pc[EX][EY];
01011
01013 float cl[EX][EY];
01014
01016 float plcl[EX][EY];
01017
01019 float plfc[EX][EY];
01020
01022 float pel[EX][EY];
01023
01025 float cape[EX][EY];
01026
01028 float z[EX][EY][EP];
01029
01031 float t[EX][EY][EP];
01032
01034 float u[EX][EY][EP];
01035
01037 float v[EX][EY][EP];
01038
01040 float w[EX][EY][EP];
01041
01043 float pv[EX][EY][EP];
01044
01046 float h2o[EX][EY][EP];
01047
01049 float o3[EX][EY][EP];
01050
01052 float lwc[EX][EY][EP];
01053
01055 float iwc[EX][EY][EP];
01056
01058 float pl[EX][EY][EP];
01059
01060 } met_t;
01061
01062 /* -----
01063    Functions...
01064    ----- */
01065
01067 void cart2geo(
01068     double *x,
01069     double *z,
01070     double *lon,
01071     double *lat);
01072
01074 #ifdef _OPENACC
01075 #pragma acc routine (check_finite)
01076 #endif
01077 int check_finite(
01078     const double x);
01079
01081 #ifdef _OPENACC
01082 #pragma acc routine (clim_hno3)
01083 #endif
01084 double clim_hno3(
01085     double t,
01086     double lat,
01087     double p);
01088
01090 #ifdef _OPENACC
01091 #pragma acc routine (clim_oh)
01092 #endif

```



```
01093 double clim_oh(
01094     double t,
01095     double lat,
01096     double p);
01097
01099 #ifdef _OPENACC
01100 #pragma acc routine (clim_tropo)
01101 #endif
01102 double clim_tropo(
01103     double t,
01104     double lat);
01105
01107 void day2doy(
01108     int year,
01109     int mon,
01110     int day,
01111     int *doy);
01112
01114 void doy2day(
01115     int year,
01116     int doy,
01117     int *mon,
01118     int *day);
01119
01121 void geo2cart(
01122     double z,
01123     double lon,
01124     double lat,
01125     double *x);
01126
01128 void get_met(
01129     ctl_t * ctl,
01130     double t,
01131     met_t ** met0,
01132     met_t ** met1);
01133
01135 void get_met_help(
01136     double t,
01137     int direct,
01138     char *metbase,
01139     double dt_met,
01140     char *filename);
01141
01143 void get_met_replace(
01144     char *orig,
01145     char *search,
01146     char *repl);
01147
01149 #ifdef _OPENACC
01150 #pragma acc routine (intpol_met_space_3d)
01151 #endif
01152 void intpol_met_space_3d(
01153     met_t * met,
01154     float array[EX][EY][EP],
01155     double p,
01156     double lon,
01157     double lat,
01158     double *var,
01159     int *ci,
01160     double *cw,
01161     int init);
01162
01164 #ifdef _OPENACC
01165 #pragma acc routine (intpol_met_space_2d)
01166 #endif
01167 void intpol_met_space_2d(
01168     met_t * met,
01169     float array[EX][EY],
01170     double lon,
01171     double lat,
01172     double *var,
01173     int *ci,
01174     double *cw,
01175     int init);
01176
01178 #ifdef _OPENACC
01179 #pragma acc routine (intpol_met_time_3d)
01180 #endif
01181 void intpol_met_time_3d(
01182     met_t * met0,
01183     float array0[EX][EY][EP],
01184     met_t * met1,
01185     float array1[EX][EY][EP],
01186     double ts,
01187     double p,
01188     double lon,
01189     double lat,
```

```
01190     double *var,
01191     int *ci,
01192     double *cw,
01193     int init);
01194
01196 #ifdef _OPENACC
01197 #pragma acc routine (intpol_met_time_2d)
01198 #endif
01199 void intpol_met_time_2d(
01200     met_t * met0,
01201     float array0[EX][EY],
01202     met_t * met1,
01203     float array1[EX][EY],
01204     double ts,
01205     double lon,
01206     double lat,
01207     double *var,
01208     int *ci,
01209     double *cw,
01210     int init);
01211
01213 void jsec2time(
01214     double jsec,
01215     int *year,
01216     int *mon,
01217     int *day,
01218     int *hour,
01219     int *min,
01220     int *sec,
01221     double *remain);
01222
01224 #ifdef _OPENACC
01225 #pragma acc routine (lapse_rate)
01226 #endif
01227 double lapse_rate(
01228     double t,
01229     double h2o);
01230
01232 #ifdef _OPENACC
01233 #pragma acc routine (locate_irr)
01234 #endif
01235 int locate_irr(
01236     double *xx,
01237     int n,
01238     double x);
01239
01241 #ifdef _OPENACC
01242 #pragma acc routine (locate_reg)
01243 #endif
01244 int locate_reg(
01245     double *xx,
01246     int n,
01247     double x);
01248
01250 #ifdef _OPENACC
01251 #pragma acc routine (nat_temperature)
01252 #endif
01253 double nat_temperature(
01254     double p,
01255     double h2o,
01256     double hno3);
01257
01259 int read_atm(
01260     const char *filename,
01261     ctl_t * ctl,
01262     atm_t * atm);
01263
01265 void read_ctl(
01266     const char *filename,
01267     int argc,
01268     char *argv[],
01269     ctl_t * ctl);
01270
01272 int read_met(
01273     ctl_t * ctl,
01274     char *filename,
01275     met_t * met);
01276
01278 void read_met_cape(
01279     met_t * met);
01280
01282 void read_met_cloud(
01283     met_t * met);
01284
01286 void read_met_detrend(
01287     ctl_t * ctl,
01288     met_t * met);
```

```
01289
01291 void read_met_extrapolate(
01292     met_t * met);
01293
01295 void read_met_geopot(
01296     ctl_t * ctl,
01297     met_t * met);
01298
01300 void read_met_grid(
01301     char *filename,
01302     int ncid,
01303     ctl_t * ctl,
01304     met_t * met);
01305
01307 int read_met_help_3d(
01308     int ncid,
01309     char *varname,
01310     char *varname2,
01311     met_t * met,
01312     float dest[EX][EY][EP],
01313     float scl);
01314
01316 int read_met_help_2d(
01317     int ncid,
01318     char *varname,
01319     char *varname2,
01320     met_t * met,
01321     float dest[EX][EY],
01322     float scl);
01323
01324 /* Read meteorological data on vertical levels. */
01325 void read_met_levels(
01326     int ncid,
01327     ctl_t * ctl,
01328     met_t * met);
01329
01331 void read_met_ml2pl(
01332     ctl_t * ctl,
01333     met_t * met,
01334     float var[EX][EY][EP]);
01335
01337 void read_met_periodic(
01338     met_t * met);
01339
01341 void read_met_pv(
01342     met_t * met);
01343
01345 void read_met_sample(
01346     ctl_t * ctl,
01347     met_t * met);
01348
01350 void read_met_surface(
01351     int ncid,
01352     met_t * met);
01353
01355 void read_met_tropo(
01356     ctl_t * ctl,
01357     met_t * met);
01358
01360 double scan_ctl(
01361     const char *filename,
01362     int argc,
01363     char *argv[],
01364     const char *varname,
01365     int aridx,
01366     const char *defvalue,
01367     char *value);
01368
01370 #ifdef _OPENACC
01371 #pragma acc routine (sedi)
01372 #endif
01373 double sedi(
01374     double p,
01375     double T,
01376     double r_p,
01377     double rho_p);
01378
01380 void spline(
01381     double *x,
01382     double *y,
01383     int n,
01384     double *x2,
01385     double *y2,
01386     int n2);
01387
01389 #ifdef _OPENACC
01390 #pragma acc routine (stddev)
```

```
01391 #endif
01392 double stddev(
01393     double *data,
01394     int n);
01395
01397 void time2jsec(
01398     int year,
01399     int mon,
01400     int day,
01401     int hour,
01402     int min,
01403     int sec,
01404     double remain,
01405     double *jsec);
01406
01408 void timer(
01409     const char *name,
01410     int output);
01411
01413 void write_atm(
01414     const char *filename,
01415     ctl_t * ctl,
01416     atm_t * atm,
01417     double t);
01418
01420 void write_csi(
01421     const char *filename,
01422     ctl_t * ctl,
01423     atm_t * atm,
01424     double t);
01425
01427 void write_ens(
01428     const char *filename,
01429     ctl_t * ctl,
01430     atm_t * atm,
01431     double t);
01432
01434 void write_grid(
01435     const char *filename,
01436     ctl_t * ctl,
01437     met_t * met0,
01438     met_t * met1,
01439     atm_t * atm,
01440     double t);
01441
01443 void write_prof(
01444     const char *filename,
01445     ctl_t * ctl,
01446     met_t * met0,
01447     met_t * met1,
01448     atm_t * atm,
01449     double t);
01450
01452 void write_sample(
01453     const char *filename,
01454     ctl_t * ctl,
01455     met_t * met0,
01456     met_t * met1,
01457     atm_t * atm,
01458     double t);
01459
01461 void write_station(
01462     const char *filename,
01463     ctl_t * ctl,
01464     atm_t * atm,
01465     double t);
```

## 5.25 met\_map.c File Reference

### Functions

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

#### 5.25.1 Detailed Description

Extract map from meteorological data.

Definition in file [met\\_map.c](#).

## 5.25.2 Function Documentation

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

Definition at line 41 of file `met_map.c`.

```
00043     {
00044
00045     ctl_t ctl;
00046
00047     met_t *met;
00048
00049     FILE *out;
00050
00051     static double timem[NX][NY], p0, ps, psm[NX][NY], ts, tsm[NX][NY],
00052         zs, zsm[NX][NY], us, usm[NX][NY], vs, vsm[NX][NY], pt, ptm[NX][NY], t,
00053         pm[NX][NY], tm[NX][NY], u, um[NX][NY], v, vm[NX][NY], w, wm[NX][NY], h2o,
00054         h2om[NX][NY], h2ot, h2otm[NX][NY], o3, o3m[NX][NY], lwc, lwcm[NX][NY],
00055         iwc, iwcm[NX][NY], z, zm[NX][NY], pv, pvm[NX][NY], zt, ztm[NX][NY],
00056         tt, ttm[NX][NY], pc, pcm[NX][NY], cl, clm[NX][NY], plcl, plclm[NX][NY],
00057         plfc, plfcm[NX][NY], pel, pelm[NX][NY], cape, capem[NX][NY], theta,
00058         ptop, pbot, t0, lon, lon0, lonl, lons[NX], dlon,
00059         lat, lat0, latl, lats[NY], dlat, cw[3];
00060
00061     static int i, ix, iy, np[NX][NY], nx, ny, ci[3];
00062
00063     /* Allocate... */
00064     ALLOC(met, met_t, 1);
00065
00066     /* Check arguments... */
00067     if (argc < 4)
00068         ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00069
00070     /* Read control parameters... */
00071     read_ctl(argv[1], argc, argv, &ctl);
00072     p0 = P(scan_ctl(argv[1], argc, argv, "MAP_Z0", -1, "10", NULL));
00073     lon0 = scan_ctl(argv[1], argc, argv, "MAP_LON0", -1, "-180", NULL);
00074     lonl = scan_ctl(argv[1], argc, argv, "MAP_LON1", -1, "180", NULL);
00075     dlon = scan_ctl(argv[1], argc, argv, "MAP_DLON", -1, "-999", NULL);
00076     lat0 = scan_ctl(argv[1], argc, argv, "MAP_LAT0", -1, "-90", NULL);
00077     latl = scan_ctl(argv[1], argc, argv, "MAP_LAT1", -1, "90", NULL);
00078     dlat = scan_ctl(argv[1], argc, argv, "MAP_DLAT", -1, "-999", NULL);
00079     theta = scan_ctl(argv[1], argc, argv, "MAP_THETA", -1, "-999", NULL);
00080
00081     /* Loop over files... */
00082     for (i = 3; i < argc; i++) {
00083
00084         /* Read meteorological data... */
00085         if (!read_met(&ctl, argv[i], met))
00086             continue;
00087
00088         /* Set horizontal grid... */
00089         if (dlon <= 0)
00090             dlon = fabs(met->lon[1] - met->lon[0]);
00091         if (dlat <= 0)
00092             dlat = fabs(met->lat[1] - met->lat[0]);
00093         if (lon0 < -360 && lonl > 360) {
00094             lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00095             lonl = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00096         }
00097         nx = ny = 0;
00098         for (lon = lon0; lon <= lonl; lon += dlon) {
00099             lons[nx] = lon;
00100             if ((++nx) > NX)
00101                 ERRMSG("Too many longitudes!");
00102         }
00103         if (lat0 < -90 && latl > 90) {
00104             lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00105             latl = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00106         }
00107         for (lat = lat0; lat <= latl; lat += dlat) {
00108             lats[ny] = lat;
00109             if ((++ny) > NY)
00110                 ERRMSG("Too many latitudes!");
00111         }
00112
00113         /* Average... */
00114         for (ix = 0; ix < nx; ix++)
```

```

00115     for (iy = 0; iy < ny; iy++) {
00116
00117         /* Find pressure level for given theta level... */
00118         if (theta > 0) {
00119             ptop = met->p[met->np - 1];
00120             pbot = met->p[0];
00121             do {
00122                 p0 = 0.5 * (ptop + pbot);
00123                 intpol_met_space_3d(met, met->t, p0, lons[ix], lats[iy],
00124                                     &t0, ci, cw, 1);
00125                 if (THETA(p0, t0) > theta)
00126                     ptop = p0;
00127                 else
00128                     pbot = p0;
00129             } while (fabs(ptop - pbot) > 1e-5);
00130         }
00131
00132         /* Interpolate meteo data... */
00133         INTPOL_SPACE_ALL(p0, lons[ix], lats[iy]);
00134
00135         /* Averaging... */
00136         timem[ix][iy] += met->time;
00137         zm[ix][iy] += z;
00138         pm[ix][iy] += p0;
00139         tm[ix][iy] += t;
00140         um[ix][iy] += u;
00141         vm[ix][iy] += v;
00142         wm[ix][iy] += w;
00143         pvm[ix][iy] += pv;
00144         h2om[ix][iy] += h2o;
00145         o3m[ix][iy] += o3;
00146         lwcm[ix][iy] += lwc;
00147         iwcm[ix][iy] += iwc;
00148         psm[ix][iy] += ps;
00149         tsm[ix][iy] += ts;
00150         zsm[ix][iy] += zs;
00151         usm[ix][iy] += us;
00152         vsm[ix][iy] += vs;
00153         ptm[ix][iy] += pt;
00154         pcm[ix][iy] += pc;
00155         clm[ix][iy] += cl;
00156         plclm[ix][iy] += plcl;
00157         plfcm[ix][iy] += plfc;
00158         pelm[ix][iy] += pel;
00159         capem[ix][iy] += cape;
00160         ztm[ix][iy] += zt;
00161         ttm[ix][iy] += tt;
00162         h2otm[ix][iy] += h2ot;
00163         np[ix][iy]++;
00164     }
00165 }
00166
00167 /* Create output file... */
00168 printf("Write meteorological data file: %s\n", argv[2]);
00169 if (!(out = fopen(argv[2], "w")))
00170     ERRMSG("Cannot create file!");
00171
00172 /* Write header... */
00173 fprintf(out,
00174         "# $1 = time [s]\n"
00175         "# $2 = altitude [km]\n"
00176         "# $3 = longitude [deg]\n"
00177         "# $4 = latitude [deg]\n"
00178         "# $5 = pressure [hPa]\n"
00179         "# $6 = temperature [K]\n"
00180         "# $7 = zonal wind [m/s]\n"
00181         "# $8 = meridional wind [m/s]\n"
00182         "# $9 = vertical velocity [hPa/s]\n"
00183         "# $10 = H2O volume mixing ratio [ppv]\n");
00184 fprintf(out,
00185         "# $11 = O3 volume mixing ratio [ppv]\n"
00186         "# $12 = geopotential height [km]\n"
00187         "# $13 = potential vorticity [PVU]\n"
00188         "# $14 = surface pressure [hPa]\n"
00189         "# $15 = surface temperature [K]\n"
00190         "# $16 = surface geopotential height [km]\n"
00191         "# $17 = surface zonal wind [m/s]\n"
00192         "# $18 = surface meridional wind [m/s]\n"
00193         "# $19 = tropopause pressure [hPa]\n"
00194         "# $20 = tropopause geopotential height [km]\n");
00195 fprintf(out,
00196         "# $21 = tropopause temperature [K]\n"
00197         "# $22 = tropopause water vapor [ppv]\n"
00198         "# $23 = cloud liquid water content [kg/kg]\n"
00199         "# $24 = cloud ice water content [kg/kg]\n"
00200         "# $25 = total column cloud water [kg/m^2]\n"
00201         "# $26 = cloud top pressure [hPa]\n"

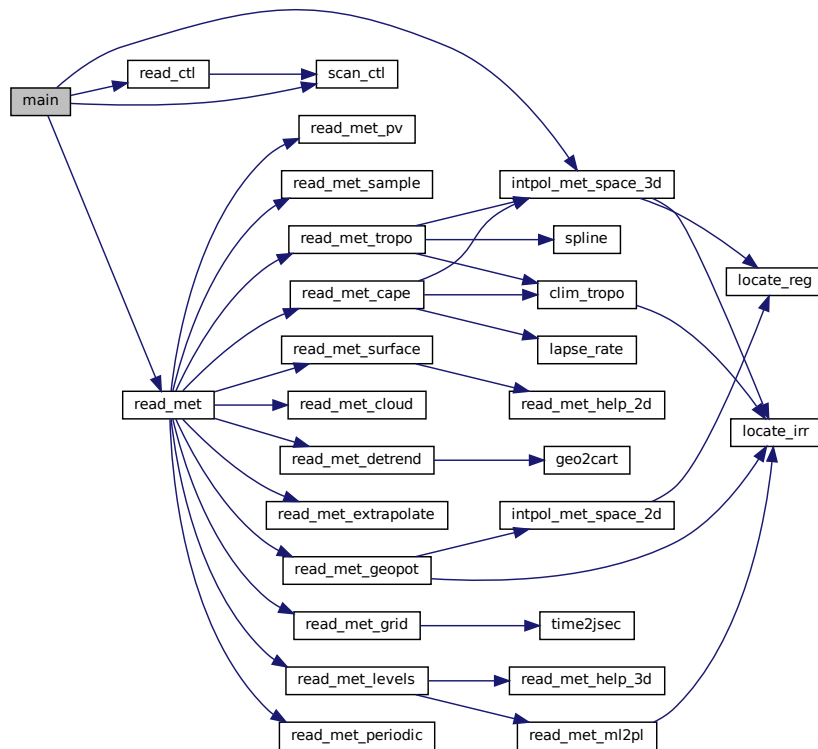
```

```

00202     "# $27 = pressure at lifted condensation level (LCL) [hPa]\n"
00203     "# $28 = pressure at level of free convection (LFC) [hPa]\n"
00204     "# $29 = pressure at equilibrium level (EL) [hPa]\n"
00205     "# $30 = convective available potential energy (CAPE) [J/kg]\n");
00206 fprintf(out,
00207     "# $31 = relative humidity over water [%%]\n"
00208     "# $32 = relative humidity over ice [%%]\n"
00209     "# $33 = dew point temperature [K]\n"
00210     "# $34 = frost point temperature [K]\n");
00211
00212 /* Write data... */
00213 for (iy = 0; iy < ny; iy++) {
00214     fprintf(out, "\n");
00215     for (ix = 0; ix < nx; ix++)
00216         fprintf(out,
00217             "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00218             " %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00219             timem[ix][iy] / np[ix][iy], Z(pm[ix][iy] / np[ix][iy]),
00220             lons[ix], lats[iy], pm[ix][iy] / np[ix][iy],
00221             tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00222             vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00223             h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00224             zm[ix][iy] / np[ix][iy], pvm[ix][iy] / np[ix][iy],
00225             psm[ix][iy] / np[ix][iy], tsm[ix][iy] / np[ix][iy],
00226             zsm[ix][iy] / np[ix][iy], usm[ix][iy] / np[ix][iy],
00227             vsm[ix][iy] / np[ix][iy], ptm[ix][iy] / np[ix][iy],
00228             ztm[ix][iy] / np[ix][iy], ttm[ix][iy] / np[ix][iy],
00229             h2otm[ix][iy] / np[ix][iy], lwcm[ix][iy] / np[ix][iy],
00230             iwcm[ix][iy] / np[ix][iy], clm[ix][iy] / np[ix][iy],
00231             pcm[ix][iy] / np[ix][iy], plclm[ix][iy] / np[ix][iy],
00232             plfcm[ix][iy] / np[ix][iy], pelm[ix][iy] / np[ix][iy],
00233             capem[ix][iy] / np[ix][iy],
00234             RH(pm[ix][iy] / np[ix][iy], tm[ix][iy] / np[ix][iy],
00235              h2om[ix][iy] / np[ix][iy]),
00236             RHICE(pm[ix][iy] / np[ix][iy], tm[ix][iy] / np[ix][iy],
00237              h2om[ix][iy] / np[ix][iy]),
00238             TDEW(pm[ix][iy] / np[ix][iy], h2om[ix][iy] / np[ix][iy]),
00239             TICE(pm[ix][iy] / np[ix][iy], h2om[ix][iy] / np[ix][iy]));
00240 }
00241
00242 /* Close file... */
00243 fclose(out);
00244
00245 /* Free... */
00246 free(met);
00247
00248 return EXIT_SUCCESS;
00249 }

```

Here is the call graph for this function:



## 5.26 met\_map.c

```

00001 /*
00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.
00008
00009  MPTRAC is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU General Public License for more details.
00013
00014  You should have received a copy of the GNU General Public License
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028  Dimensions...
00029  ----- */
00030
00032 #define NX 1441
00033
00035 #define NY 721
00036
00037 /* -----
00038  Main...
00039  ----- */
00040
00041 int main(
00042     int argc,
00043     char *argv[]) {
00044

```



```

00045     ctl_t ctl;
00046
00047     met_t *met;
00048
00049     FILE *out;
00050
00051     static double timem[NX][NY], p0, ps, psm[NX][NY], ts, tsm[NX][NY],
00052         zs, zsm[NX][NY], us, usm[NX][NY], vs, vsm[NX][NY], pt, ptm[NX][NY], t,
00053         pm[NX][NY], tm[NX][NY], u, um[NX][NY], v, vm[NX][NY], w, wm[NX][NY], h2o,
00054         h2om[NX][NY], h2ot, h2otm[NX][NY], o3, o3m[NX][NY], lwc, lwc[NX][NY],
00055         iwc, iwc[NX][NY], z, zm[NX][NY], pv, pvm[NX][NY], zt, ztm[NX][NY],
00056         tt, ttm[NX][NY], pc, pcm[NX][NY], cl, clm[NX][NY], plcl, plclm[NX][NY],
00057         plfc, plfc[NX][NY], pel, pelm[NX][NY], cape, capem[NX][NY], theta,
00058         ptop, pbot, t0, lon, lon0, lon1, lons[NX], dlon,
00059         lat, lat0, lat1, lats[NY], dlat, cw[3];
00060
00061     static int i, ix, iy, np[NX][NY], nx, ny, ci[3];
00062
00063     /* Allocate... */
00064     ALLOC(met, met_t, 1);
00065
00066     /* Check arguments... */
00067     if (argc < 4)
00068         ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00069
00070     /* Read control parameters... */
00071     read_ctl(argv[1], argc, argv, &ctl);
00072     p0 = P(scan_ctl(argv[1], argc, argv, "MAP_Z0", -1, "10", NULL));
00073     lon0 = scan_ctl(argv[1], argc, argv, "MAP_LON0", -1, "-180", NULL);
00074     lon1 = scan_ctl(argv[1], argc, argv, "MAP_LON1", -1, "180", NULL);
00075     dlon = scan_ctl(argv[1], argc, argv, "MAP_DLON", -1, "-999", NULL);
00076     lat0 = scan_ctl(argv[1], argc, argv, "MAP_LAT0", -1, "-90", NULL);
00077     lat1 = scan_ctl(argv[1], argc, argv, "MAP_LAT1", -1, "90", NULL);
00078     dlat = scan_ctl(argv[1], argc, argv, "MAP_DLAT", -1, "-999", NULL);
00079     theta = scan_ctl(argv[1], argc, argv, "MAP_THETA", -1, "-999", NULL);
00080
00081     /* Loop over files... */
00082     for (i = 3; i < argc; i++) {
00083
00084         /* Read meteorological data... */
00085         if (!read_met(&ctl, argv[i], met))
00086             continue;
00087
00088         /* Set horizontal grid... */
00089         if (dlon <= 0)
00090             dlon = fabs(met->lon[1] - met->lon[0]);
00091         if (dlat <= 0)
00092             dlat = fabs(met->lat[1] - met->lat[0]);
00093         if (lon0 < -360 && lon1 > 360) {
00094             lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00095             lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00096         }
00097         nx = ny = 0;
00098         for (lon = lon0; lon <= lon1; lon += dlon) {
00099             lons[nx] = lon;
00100             if ((++nx) > NX)
00101                 ERRMSG("Too many longitudes!");
00102         }
00103         if (lat0 < -90 && lat1 > 90) {
00104             lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00105             lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00106         }
00107         for (lat = lat0; lat <= lat1; lat += dlat) {
00108             lats[ny] = lat;
00109             if ((++ny) > NY)
00110                 ERRMSG("Too many latitudes!");
00111         }
00112
00113         /* Average... */
00114         for (ix = 0; ix < nx; ix++)
00115             for (iy = 0; iy < ny; iy++) {
00116
00117                 /* Find pressure level for given theta level... */
00118                 if (theta > 0) {
00119                     ptop = met->p[met->np - 1];
00120                     pbot = met->p[0];
00121                     do {
00122                         p0 = 0.5 * (ptop + pbot);
00123                         interpol_met_space_3d(met, met->t, p0, lons[ix], lats[iy],
00124                             &t0, ci, cw, 1);
00125                         if (THETA(p0, t0) > theta)
00126                             ptop = p0;
00127                         else
00128                             pbot = p0;
00129                     } while (fabs(ptop - pbot) > 1e-5);
00130                 }
00131             }

```

```

00132      /* Interpolate meteo data... */
00133      INTPOL_SPACE_ALL(p0, lons[ix], lats[iy]);
00134
00135      /* Averaging... */
00136      timem[ix][iy] += met->time;
00137      zm[ix][iy] += z;
00138      pm[ix][iy] += p0;
00139      tm[ix][iy] += t;
00140      um[ix][iy] += u;
00141      vm[ix][iy] += v;
00142      wm[ix][iy] += w;
00143      pvm[ix][iy] += pv;
00144      h2om[ix][iy] += h2o;
00145      o3m[ix][iy] += o3;
00146      lwcm[ix][iy] += lwc;
00147      iwcm[ix][iy] += iwc;
00148      psm[ix][iy] += ps;
00149      tsm[ix][iy] += ts;
00150      zsm[ix][iy] += zs;
00151      usm[ix][iy] += us;
00152      vsm[ix][iy] += vs;
00153      ptm[ix][iy] += pt;
00154      pcm[ix][iy] += pc;
00155      clm[ix][iy] += cl;
00156      plclm[ix][iy] += plcl;
00157      plfcm[ix][iy] += plfc;
00158      pelm[ix][iy] += pel;
00159      capem[ix][iy] += cape;
00160      ztm[ix][iy] += zt;
00161      ttm[ix][iy] += tt;
00162      h2otm[ix][iy] += h2ot;
00163      np[ix][iy]++;
00164    }
00165  }
00166
00167  /* Create output file... */
00168  printf("Write meteorological data file: %s\n", argv[2]);
00169  if (!(out = fopen(argv[2], "w")))
00170    ERRMSG("Cannot create file!");
00171
00172  /* Write header... */
00173  fprintf(out,
00174    "# $1 = time [s]\n"
00175    "# $2 = altitude [km]\n"
00176    "# $3 = longitude [deg]\n"
00177    "# $4 = latitude [deg]\n"
00178    "# $5 = pressure [hPa]\n"
00179    "# $6 = temperature [K]\n"
00180    "# $7 = zonal wind [m/s]\n"
00181    "# $8 = meridional wind [m/s]\n"
00182    "# $9 = vertical velocity [hPa/s]\n"
00183    "# $10 = H2O volume mixing ratio [ppv]\n");
00184  fprintf(out,
00185    "# $11 = O3 volume mixing ratio [ppv]\n"
00186    "# $12 = geopotential height [km]\n"
00187    "# $13 = potential vorticity [PVU]\n"
00188    "# $14 = surface pressure [hPa]\n"
00189    "# $15 = surface temperature [K]\n"
00190    "# $16 = surface geopotential height [km]\n"
00191    "# $17 = surface zonal wind [m/s]\n"
00192    "# $18 = surface meridional wind [m/s]\n"
00193    "# $19 = tropopause pressure [hPa]\n"
00194    "# $20 = tropopause geopotential height [km]\n");
00195  fprintf(out,
00196    "# $21 = tropopause temperature [K]\n"
00197    "# $22 = tropopause water vapor [ppv]\n"
00198    "# $23 = cloud liquid water content [kg/kg]\n"
00199    "# $24 = cloud ice water content [kg/kg]\n"
00200    "# $25 = total column cloud water [kg/m^2]\n"
00201    "# $26 = cloud top pressure [hPa]\n"
00202    "# $27 = pressure at lifted condensation level (LCL) [hPa]\n"
00203    "# $28 = pressure at level of free convection (LFC) [hPa]\n"
00204    "# $29 = pressure at equilibrium level (EL) [hPa]\n"
00205    "# $30 = convective available potential energy (CAPE) [J/kg]\n");
00206  fprintf(out,
00207    "# $31 = relative humidity over water [%%]\n"
00208    "# $32 = relative humidity over ice [%%]\n"
00209    "# $33 = dew point temperature [K]\n"
00210    "# $34 = frost point temperature [K]\n");
00211
00212  /* Write data... */
00213  for (iy = 0; iy < ny; iy++) {
00214    fprintf(out, "\n");
00215    for (ix = 0; ix < nx; ix++)
00216      fprintf(out,
00217        "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00218        " %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",

```

```

00219         timem[ix][iy] / np[ix][iy], Z(pm[ix][iy] / np[ix][iy]),
00220         lons[ix], lats[iy], pm[ix][iy] / np[ix][iy],
00221         tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00222         vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00223         h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00224         zm[ix][iy] / np[ix][iy], pvm[ix][iy] / np[ix][iy],
00225         psm[ix][iy] / np[ix][iy], tsm[ix][iy] / np[ix][iy],
00226         zsm[ix][iy] / np[ix][iy], usm[ix][iy] / np[ix][iy],
00227         vsm[ix][iy] / np[ix][iy], ptm[ix][iy] / np[ix][iy],
00228         ztm[ix][iy] / np[ix][iy], ttm[ix][iy] / np[ix][iy],
00229         h2otm[ix][iy] / np[ix][iy], lwcm[ix][iy] / np[ix][iy],
00230         iwc[ix][iy] / np[ix][iy], clm[ix][iy] / np[ix][iy],
00231         pcm[ix][iy] / np[ix][iy], plclm[ix][iy] / np[ix][iy],
00232         plfcm[ix][iy] / np[ix][iy], pelm[ix][iy] / np[ix][iy],
00233         capem[ix][iy] / np[ix][iy],
00234         RH(pm[ix][iy] / np[ix][iy], tm[ix][iy] / np[ix][iy],
00235            h2om[ix][iy] / np[ix][iy]),
00236         RHICE(pm[ix][iy] / np[ix][iy], tm[ix][iy] / np[ix][iy],
00237            h2om[ix][iy] / np[ix][iy]),
00238         TDEW(pm[ix][iy] / np[ix][iy], h2om[ix][iy] / np[ix][iy]),
00239         TICE(pm[ix][iy] / np[ix][iy], h2om[ix][iy] / np[ix][iy]));
00240     }
00241
00242     /* Close file... */
00243     fclose(out);
00244
00245     /* Free... */
00246     free(met);
00247
00248     return EXIT_SUCCESS;
00249 }

```

## 5.27 met\_prof.c File Reference

### Functions

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

#### 5.27.1 Detailed Description

Extract vertical profile from meteorological data.

Definition in file [met\\_prof.c](#).

#### 5.27.2 Function Documentation

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

Definition at line 38 of file [met\\_prof.c](#).

```

00040     {
00041
00042     ctl_t ctl;
00043
00044     met_t *met;
00045
00046     FILE *out;
00047
00048     static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
00049         lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ], w,
00050         wm[NZ], h2o, h2om[NZ], h2ot, h2otm[NZ], o3, o3m[NZ], lwc, lwcm[NZ],
00051         iwc, iwc[NZ], ps, psm[NZ], ts, tsm[NZ], zs, zsm[NZ], us, usm[NZ],
00052         vs, vsm[NZ], pt, ptm[NZ], pc, pcm[NZ], cl, clm[NZ],

```

```

00053     plcl, plclm[NZ], plfc, plfcm[NZ], pel, pelm[NZ], cape, capem[NZ],
00054     tt, ttm[NZ], zm[NZ], zt, ztm[NZ], pv, pvm[NZ], plev[NZ], cw[3];
00055
00056     static int i, iz, np[NZ], npt[NZ], nz, ci[3];
00057
00058     /* Allocate... */
00059     ALLOC(met, met_t, 1);
00060
00061     /* Check arguments... */
00062     if (argc < 4)
00063         ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00064
00065     /* Read control parameters... */
00066     read_ctl(argv[1], argc, argv, &ctl);
00067     z0 = scan_ctl(argv[1], argc, argv, "PROF_Z0", -1, "-999", NULL);
00068     z1 = scan_ctl(argv[1], argc, argv, "PROF_Z1", -1, "-999", NULL);
00069     dz = scan_ctl(argv[1], argc, argv, "PROF_DZ", -1, "-999", NULL);
00070     lon0 = scan_ctl(argv[1], argc, argv, "PROF_LON0", -1, "0", NULL);
00071     lon1 = scan_ctl(argv[1], argc, argv, "PROF_LON1", -1, "0", NULL);
00072     dlon = scan_ctl(argv[1], argc, argv, "PROF_DLON", -1, "-999", NULL);
00073     lat0 = scan_ctl(argv[1], argc, argv, "PROF_LAT0", -1, "0", NULL);
00074     lat1 = scan_ctl(argv[1], argc, argv, "PROF_LAT1", -1, "0", NULL);
00075     dlat = scan_ctl(argv[1], argc, argv, "PROF_DLAT", -1, "-999", NULL);
00076
00077     /* Loop over input files... */
00078     for (i = 3; i < argc; i++) {
00079
00080         /* Read meteorological data... */
00081         if (!read_met(&ctl, argv[i], met))
00082             continue;
00083
00084         /* Set vertical grid... */
00085         if (z0 < 0)
00086             z0 = Z(met->p[0]);
00087         if (z1 < 0)
00088             z1 = Z(met->p[met->np - 1]);
00089         nz = 0;
00090         if (dz < 0) {
00091             for (iz = 0; iz < met->np; iz++)
00092                 if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00093                     plev[nz] = met->p[iz];
00094                     if ((++nz) > NZ)
00095                         ERRMSG("Too many pressure levels!");
00096                 }
00097         } else
00098             for (z = z0; z <= z1; z += dz) {
00099                 plev[nz] = P(z);
00100                 if ((++nz) > NZ)
00101                     ERRMSG("Too many pressure levels!");
00102             }
00103
00104         /* Set horizontal grid... */
00105         if (dlon <= 0)
00106             dlon = fabs(met->lon[1] - met->lon[0]);
00107         if (dlat <= 0)
00108             dlat = fabs(met->lat[1] - met->lat[0]);
00109
00110         /* Average... */
00111         for (iz = 0; iz < nz; iz++)
00112             for (lon = lon0; lon <= lon1; lon += dlon)
00113                 for (lat = lat0; lat <= lat1; lat += dlat) {
00114
00115                     /* Interpolate meteo data... */
00116                     INTPOL_SPACE_ALL(plev[iz], lon, lat);
00117
00118                     /* Averaging... */
00119                     if (gsl_finite(t) && gsl_finite(u)
00120                         && gsl_finite(v) && gsl_finite(w)) {
00121                         timem[iz] += met->time;
00122                         lonm[iz] += lon;
00123                         latm[iz] += lat;
00124                         zm[iz] += z;
00125                         tm[iz] += t;
00126                         um[iz] += u;
00127                         vm[iz] += v;
00128                         wm[iz] += w;
00129                         pvm[iz] += pv;
00130                         h2om[iz] += h2o;
00131                         o3m[iz] += o3;
00132                         psm[iz] += ps;
00133                         tsm[iz] += ts;
00134                         zsm[iz] += zs;
00135                         usm[iz] += us;
00136                         vsm[iz] += vs;
00137                         pcm[iz] += pc;
00138                         clm[iz] += cl;
00139                         plclm[iz] += plcl;

```

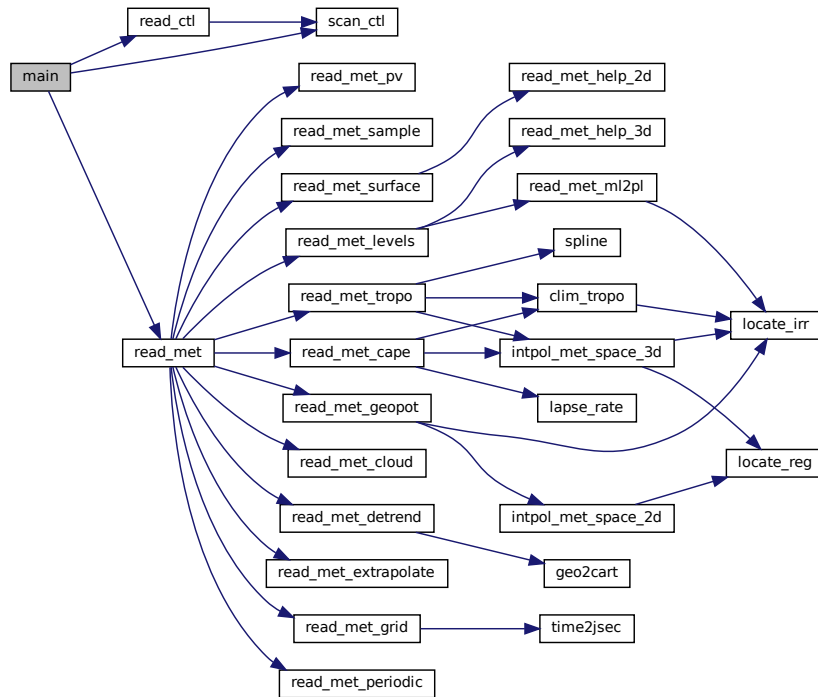
Generated by Doxygen

```

00227
00228     return EXIT_SUCCESS;
00229 }

```

Here is the call graph for this function:



## 5.28 met\_prof.c

```

00001 /*
00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.
00008
00009  MPTRAC is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU General Public License for more details.
00013
00014  You should have received a copy of the GNU General Public License
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00020 #include "libtrac.h"
00021
00022 /* -----
00023  Dimensions...
00024  ----- */
00025
00026 #define NZ 1000
00027
00028 /* -----
00029  Main...
00030  ----- */
00031
00032 int main(
00033     int argc,
00034     char *argv[]) {

```

```

00041
00042     ctl_t ctl;
00043
00044     met_t *met;
00045
00046     FILE *out;
00047
00048     static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
00049         lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ], w,
00050         wm[NZ], h2o, h2om[NZ], h2ot, h2otm[NZ], o3, o3m[NZ], lwc, lwcm[NZ],
00051         iwc, iwcm[NZ], ps, psm[NZ], ts, tsm[NZ], zs, zsm[NZ], us, usm[NZ],
00052         vs, vsm[NZ], pt, ptm[NZ], pc, pcm[NZ], cl, clm[NZ],
00053         plcl, plclm[NZ], plfc, plfcm[NZ], pel, pelm[NZ], cape, capem[NZ],
00054         tt, ttm[NZ], zm[NZ], zt, ztm[NZ], pv, pvm[NZ], plev[NZ], cw[3];
00055
00056     static int i, iz, np[NZ], npt[NZ], nz, ci[3];
00057
00058     /* Allocate... */
00059     ALLOC(met, met_t, 1);
00060
00061     /* Check arguments... */
00062     if (argc < 4)
00063         ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00064
00065     /* Read control parameters... */
00066     read_ctl(argv[1], argc, argv, &ctl);
00067     z0 = scan_ctl(argv[1], argc, argv, "PROF_Z0", -1, "-999", NULL);
00068     z1 = scan_ctl(argv[1], argc, argv, "PROF_Z1", -1, "-999", NULL);
00069     dz = scan_ctl(argv[1], argc, argv, "PROF_DZ", -1, "-999", NULL);
00070     lon0 = scan_ctl(argv[1], argc, argv, "PROF_LON0", -1, "0", NULL);
00071     lon1 = scan_ctl(argv[1], argc, argv, "PROF_LON1", -1, "0", NULL);
00072     dlon = scan_ctl(argv[1], argc, argv, "PROF_DLON", -1, "-999", NULL);
00073     lat0 = scan_ctl(argv[1], argc, argv, "PROF_LAT0", -1, "0", NULL);
00074     lat1 = scan_ctl(argv[1], argc, argv, "PROF_LAT1", -1, "0", NULL);
00075     dlat = scan_ctl(argv[1], argc, argv, "PROF_DLAT", -1, "-999", NULL);
00076
00077     /* Loop over input files... */
00078     for (i = 3; i < argc; i++) {
00079
00080         /* Read meteorological data... */
00081         if (!read_met(&ctl, argv[i], met))
00082             continue;
00083
00084         /* Set vertical grid... */
00085         if (z0 < 0)
00086             z0 = Z(met->p[0]);
00087         if (z1 < 0)
00088             z1 = Z(met->p[met->np - 1]);
00089         nz = 0;
00090         if (dz < 0) {
00091             for (iz = 0; iz < met->np; iz++)
00092                 if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00093                     plev[nz] = met->p[iz];
00094                     if ((++nz) > NZ)
00095                         ERRMSG("Too many pressure levels!");
00096                 }
00097             } else
00098                 for (z = z0; z <= z1; z += dz) {
00099                     plev[nz] = P(z);
00100                     if ((++nz) > NZ)
00101                         ERRMSG("Too many pressure levels!");
00102                 }
00103
00104         /* Set horizontal grid... */
00105         if (dlon <= 0)
00106             dlon = fabs(met->lon[1] - met->lon[0]);
00107         if (dlat <= 0)
00108             dlat = fabs(met->lat[1] - met->lat[0]);
00109
00110         /* Average... */
00111         for (iz = 0; iz < nz; iz++)
00112             for (lon = lon0; lon <= lon1; lon += dlon)
00113                 for (lat = lat0; lat <= lat1; lat += dlat) {
00114
00115                     /* Interpolate meteo data... */
00116                     INTPOL_SPACE_ALL(plev[iz], lon, lat);
00117
00118                     /* Averaging... */
00119                     if (gsl_finite(t) && gsl_finite(u)
00120                         && gsl_finite(v) && gsl_finite(w)) {
00121                         timem[iz] += met->time;
00122                         lonm[iz] += lon;
00123                         latm[iz] += lat;
00124                         zm[iz] += z;
00125                         tm[iz] += t;
00126                         um[iz] += u;
00127                         vm[iz] += v;

```

```
00128      wm[iiz] += w;
00129      pvm[iiz] += pv;
00130      h2om[iiz] += h2o;
00131      o3m[iiz] += o3;
00132      psm[iiz] += ps;
00133      tsm[iiz] += ts;
00134      zsm[iiz] += zs;
00135      usm[iiz] += us;
00136      vsm[iiz] += vs;
00137      pcm[iiz] += pc;
00138      clm[iiz] += cl;
00139      plclm[iiz] += plcl;
00140      plfcm[iiz] += plfc;
00141      pelm[iiz] += pel;
00142      capem[iiz] += cape;
00143      lwcm[iiz] += lwc;
00144      iwcm[iiz] += iwc;
00145      if (gsl_finite(pt)) {
00146          ptm[iiz] += pt;
00147          ztm[iiz] += zt;
00148          ttM[iiz] += tt;
00149          h2otm[iiz] += h2ot;
00150          npt[iiz]++;
00151      }
00152      np[iiz]++;
00153  }
00154  }
00155  }
00156
00157  /* Create output file... */
00158  printf("Write meteorological data file: %s\n", argv[2]);
00159  if (!out = fopen(argv[2], "w"))
00160      ERRMSG("Cannot create file!");
00161
00162  /* Write header... */
00163  fprintf(out,
00164    "# $1 = time [s]\n"
00165    "# $2 = altitude [km]\n"
00166    "# $3 = longitude [deg]\n"
00167    "# $4 = latitude [deg]\n"
00168    "# $5 = pressure [hPa]\n"
00169    "# $6 = temperature [K]\n"
00170    "# $7 = zonal wind [m/s]\n"
00171    "# $8 = meridional wind [m/s]\n"
00172    "# $9 = vertical velocity [hPa/s]\n"
00173    "# $10 = H2O volume mixing ratio [ppv]\n");
00174  fprintf(out,
00175    "# $11 = O3 volume mixing ratio [ppv]\n"
00176    "# $12 = geopotential height [km]\n"
00177    "# $13 = potential vorticity [PVU]\n"
00178    "# $14 = surface pressure [hPa]\n"
00179    "# $15 = surface temperature [K]\n"
00180    "# $16 = surface geopotential height [km]\n"
00181    "# $17 = surface zonal wind [m/s]\n"
00182    "# $18 = surface meridional wind [m/s]\n"
00183    "# $19 = tropopause pressure [hPa]\n"
00184    "# $20 = tropopause geopotential height [km]\n");
00185  fprintf(out,
00186    "# $21 = tropopause temperature [K]\n"
00187    "# $22 = tropopause water vapor [ppv]\n"
00188    "# $23 = cloud liquid water content [kg/kg]\n"
00189    "# $24 = cloud ice water content [kg/kg]\n"
00190    "# $25 = total column cloud water [kg/m^2]\n"
00191    "# $26 = cloud top pressure [hPa]\n"
00192    "# $27 = pressure at lifted condensation level (LCL) [hPa]\n"
00193    "# $28 = pressure at level of free convection (LFC) [hPa]\n"
00194    "# $29 = pressure at equilibrium level (EL) [hPa]\n"
00195    "# $30 = convective available potential energy (CAPE) [J/kg]\n");
00196  fprintf(out,
00197    "# $31 = relative humidity over water [%%]\n"
00198    "# $32 = relative humidity over ice [%%]\n"
00199    "# $33 = dew point temperature [K]\n"
00200    "# $34 = frost point temperature [K]\n");
00201
00202  /* Write data... */
00203  for (iiz = 0; iiz < nz; iiz++)
00204      fprintf(out,
00205        "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00206        timem[iiz] / np[iiz], Z(plev[iiz]), lonm[iiz] / np[iiz],
00207        latm[iiz] / np[iiz], plev[iiz], tm[iiz] / np[iiz], um[iiz] / np[iiz],
00208        vm[iiz] / np[iiz], wm[iiz] / np[iiz], h2om[iiz] / np[iiz],
00209        o3m[iiz] / np[iiz], zm[iiz] / np[iiz], pvm[iiz] / np[iiz],
00210        psm[iiz] / np[iiz], tsm[iiz] / np[iiz], zsm[iiz] / np[iiz],
00211        usm[iiz] / np[iiz], vsm[iiz] / np[iiz], ptm[iiz] / npt[iiz],
00212        ztm[iiz] / npt[iiz], ttM[iiz] / npt[iiz], h2otm[iiz] / npt[iiz],
00213        lwcm[iiz] / np[iiz], iwcm[iiz] / np[iiz], clm[iiz] / np[iiz],
```



```

00215         pcm[iz] / np[iz], plclm[iz] / np[iz], plfcm[iz] / np[iz],
00216         pelm[iz] / np[iz], capem[iz] / np[iz],
00217         RH(plev[iz], tm[iz] / np[iz], h2om[iz] / np[iz]),
00218         RHICE(plev[iz], tm[iz] / np[iz], h2om[iz] / np[iz]),
00219         TDEW(plev[iz], h2om[iz] / np[iz]),
00220         TICE(plev[iz], h2om[iz] / np[iz]));
00221
00222     /* Close file... */
00223     fclose(out);
00224
00225     /* Free... */
00226     free(met);
00227
00228     return EXIT_SUCCESS;
00229 }

```

## 5.29 met\_sample.c File Reference

### Functions

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

#### 5.29.1 Detailed Description

Sample meteorological data at given geolocations.

Definition in file [met\\_sample.c](#).

#### 5.29.2 Function Documentation

**5.29.2.1 main()** 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 h2o, h2ot, o3, lwc, iwc, p0, pl, pref, ps, ts, zs, us, vs, pt, pc,
00044             cl, plcl, plfc, pel, cape, pv, t, tt, u, v, w, z, zm, zref, zt, cw[3],
00045             time_old = -999, p_old = -999, lon_old = -999, lat_old = -999;
00046
00047         int geopot, grid_time, grid_z, grid_lon, grid_lat, ip, it, ci[3];
00048
00049         /* Check arguments... */
00050         if (argc < 3)
00051             ERRMSG("Give parameters: <ctl> <sample.tab> <atm_in>");
00052
00053         /* Allocate... */
00054         ALLOC(atm, atm_t, 1);
00055         ALLOC(met0, met_t, 1);
00056         ALLOC(met1, met_t, 1);
00057
00058         /* Read control parameters... */
00059         read_ctl(argv[1], argc, argv, &ctl);
00060         geopot =
00061             (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GEOPOT", -1, "0", NULL);

```

```

00062 grid_time =
00063     (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_TIME", -1, "0", NULL);
00064 grid_z =
00065     (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_Z", -1, "0", NULL);
00066 grid_lon =
00067     (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LON", -1, "0", NULL);
00068 grid_lat =
00069     (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LAT", -1, "0", NULL);
00070
00071 /* Read atmospheric data... */
00072 if (!read_atm(argv[3], &ctl, atm))
00073     ERRMSG("Cannot open file!");
00074
00075 /* Create output file... */
00076 printf("Write meteorological data file: %s\n", argv[2]);
00077 if (!(out = fopen(argv[2], "w")))
00078     ERRMSG("Cannot create file!");
00079
00080 /* Write header... */
00081 fprintf(out,
00082     "# $1 = time [s]\n"
00083     "# $2 = altitude [km]\n"
00084     "# $3 = longitude [deg]\n"
00085     "# $4 = latitude [deg]\n"
00086     "# $5 = pressure [hPa]\n"
00087     "# $6 = temperature [K]\n"
00088     "# $7 = zonal wind [m/s]\n"
00089     "# $8 = meridional wind [m/s]\n"
00090     "# $9 = vertical velocity [hPa/s]\n"
00091     "# $10 = H2O volume mixing ratio [ppv]\n");
00092 fprintf(out,
00093     "# $11 = O3 volume mixing ratio [ppv]\n"
00094     "# $12 = geopotential height [km]\n"
00095     "# $13 = potential vorticity [PVU]\n"
00096     "# $14 = surface pressure [hPa]\n"
00097     "# $15 = surface temperature [K]\n"
00098     "# $16 = surface geopotential height [km]\n"
00099     "# $17 = surface zonal wind [m/s]\n"
00100     "# $18 = surface meridional wind [m/s]\n"
00101     "# $19 = tropopause pressure [hPa]\n"
00102     "# $20 = tropopause geopotential height [km]\n");
00103 fprintf(out,
00104     "# $21 = tropopause temperature [K]\n"
00105     "# $22 = tropopause water vapor [ppv]\n"
00106     "# $23 = cloud liquid water content [kg/kg]\n"
00107     "# $24 = cloud ice water content [kg/kg]\n"
00108     "# $25 = total column cloud water [kg/m^2]\n"
00109     "# $26 = cloud top pressure [hPa]\n"
00110     "# $27 = pressure at lifted condensation level (LCL) [hPa]\n"
00111     "# $28 = pressure at level of free convection (LFC) [hPa]\n"
00112     "# $29 = pressure at equilibrium level (EL) [hPa]\n"
00113     "# $30 = convective available potential energy (CAPE) [J/kg]\n");
00114 fprintf(out,
00115     "# $31 = relative humidity over water [%]\n"
00116     "# $32 = relative humidity over ice [%]\n"
00117     "# $33 = dew point temperature [K]\n"
00118     "# $34 = frost point temperature [K]\n");
00119
00120 /* Loop over air parcels... */
00121 for (ip = 0; ip < atm->np; ip++) {
00122
00123     /* Get meteorological data... */
00124     get_met(&ctl, atm->time[ip], &met0, &met1);
00125
00126     /* Set reference pressure for interpolation... */
00127     pref = atm->p[ip];
00128     if (geopot) {
00129         zref = Z(pref);
00130         p0 = met0->p[0];
00131         p1 = met0->p[met0->np - 1];
00132         for (it = 0; it < 24; it++) {
00133             pref = 0.5 * (p0 + p1);
00134             intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->time[ip], pref,
00135                 atm->lon[ip], atm->lat[ip], &zm, ci, cw, 1);
00136             if (zref > zm || !gsl_finite(zm))
00137                 p0 = pref;
00138             else
00139                 p1 = pref;
00140         }
00141         pref = 0.5 * (p0 + p1);
00142     }
00143
00144     /* Interpolate meteo data... */
00145     INTPOL_TIME_ALL(atm->time[ip], pref, atm->lon[ip], atm->lat[ip]);
00146
00147     /* Make blank lines... */
00148     if (ip == 0 || (grid_time && atm->time[ip] != time_old)

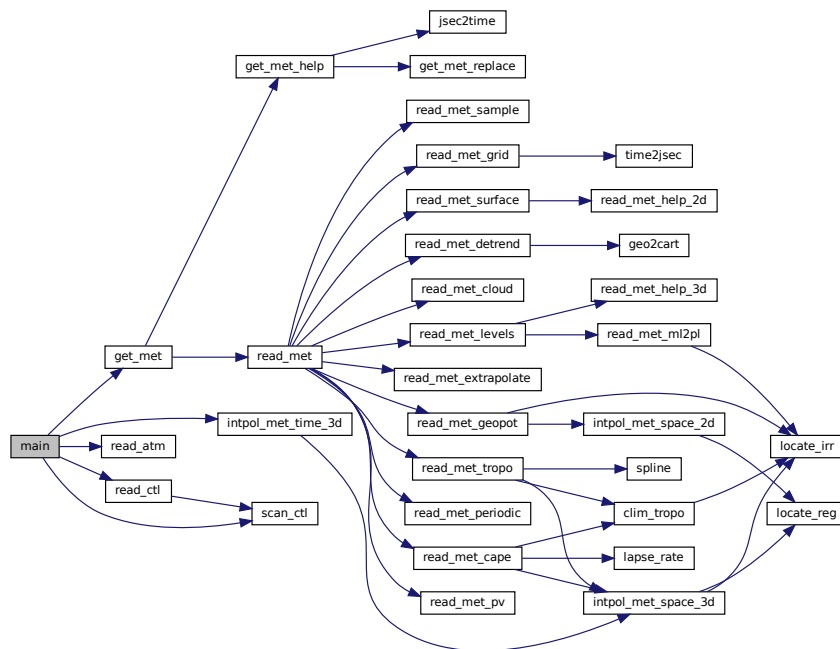
```

```

00149     || (grid_z && atm->p[ip] != p_old)
00150     || (grid_lon && atm->lon[ip] != lon_old)
00151     || (grid_lat && atm->lat[ip] != lat_old))
00152     fprintf(out, "\n");
00153     time_old = atm->time[ip];
00154     p_old = atm->p[ip];
00155     lon_old = atm->lon[ip];
00156     lat_old = atm->lat[ip];
00157
00158     /* Write data... */
00159     fprintf(out,
00160         "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00161         " %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00162         atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
00163         atm->p[ip], t, u, v, w, h2o, o3, z, pv, ps, ts, zs, us, vs,
00164         pt, zt, tt, h2ot, lwc, iwc, cl, pc, plcl, plfc, pel, cape,
00165         RH(atm->p[ip], t, h2o), RHICE(atm->p[ip], t, h2o),
00166         TDEW(atm->p[ip], h2o), TICE(atm->p[ip], h2o));
00167 }
00168
00169 /* Close file... */
00170 fclose(out);
00171
00172 /* Free... */
00173 free(atm);
00174 free(met0);
00175 free(met1);
00176
00177 return EXIT_SUCCESS;
00178 }

```

Here is the call graph for this function:



## 5.30 met\_sample.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
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028     Main...
00029 ----- */
00030
00031 int main(
00032     int argc,
00033     char *argv[]) {
00034
00035     ctl_t ctl;
00036
00037     atm_t *atm;
00038
00039     met_t *met0, *met1;
00040
00041     FILE *out;
00042
00043     double h2o, h2ot, o3, lwc, iwc, p0, p1, pref, ps, ts, zs, us, vs, pt, pc,
00044         cl, plcl, plfc, pel, cape, pv, t, tt, u, v, w, z, zm, zref, zt, cw[3],
00045         time_old = -999, p_old = -999, lon_old = -999, lat_old = -999;
00046
00047     int geopot, grid_time, grid_z, grid_lon, grid_lat, ip, it, ci[3];
00048
00049     /* Check arguments... */
00050     if (argc < 3)
00051         ERRMSG("Give parameters: <ctl> <sample.tab> <atm_in>");
00052
00053     /* Allocate... */
00054     ALLOC(atm, atm_t, 1);
00055     ALLOC(met0, met_t, 1);
00056     ALLOC(met1, met_t, 1);
00057
00058     /* Read control parameters... */
00059     read_ctl(argv[1], argc, argv, &ctl);
00060     geopot =
00061         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GEOPOT", -1, "0", NULL);
00062     grid_time =
00063         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_TIME", -1, "0", NULL);
00064     grid_z =
00065         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_Z", -1, "0", NULL);
00066     grid_lon =
00067         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LON", -1, "0", NULL);
00068     grid_lat =
00069         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LAT", -1, "0", NULL);
00070
00071     /* Read atmospheric data... */
00072     if (!read_atm(argv[3], &ctl, atm))
00073         ERRMSG("Cannot open file!");
00074
00075     /* Create output file... */
00076     printf("Write meteorological data file: %s\n", argv[2]);
00077     if (!(out = fopen(argv[2], "w")))
00078         ERRMSG("Cannot create file!");
00079
00080     /* Write header... */
00081     fprintf(out,
00082         "# $1 = time [s]\n"
00083         "# $2 = altitude [km]\n"
00084         "# $3 = longitude [deg]\n"
00085         "# $4 = latitude [deg]\n"
00086         "# $5 = pressure [hPa]\n"
00087         "# $6 = temperature [K]\n"
00088         "# $7 = zonal wind [m/s]\n"
00089         "# $8 = meridional wind [m/s]\n"
00090         "# $9 = vertical velocity [hPa/s]\n"
00091         "# $10 = H2O volume mixing ratio [ppv]\n");
00092     fprintf(out,
00093         "# $11 = O3 volume mixing ratio [ppv]\n"
00094         "# $12 = geopotential height [km]\n"
00095         "# $13 = potential vorticity [PVU]\n"
00096         "# $14 = surface pressure [hPa]\n"
00097         "# $15 = surface temperature [K]\n"
00098         "# $16 = surface geopotential height [km]\n"
00099         "# $17 = surface zonal wind [m/s]\n"
00100         "# $18 = surface meridional wind [m/s]\n"
00101         "# $19 = tropopause pressure [hPa]\n"
00102         "# $20 = tropopause geopotential height [km]\n");
00103     fprintf(out,

```

```

00104         "# $21 = tropopause temperature [K]\n"
00105         "# $22 = tropopause water vapor [ppv]\n"
00106         "# $23 = cloud liquid water content [kg/kg]\n"
00107         "# $24 = cloud ice water content [kg/kg]\n"
00108         "# $25 = total column cloud water [kg/m^2]\n"
00109         "# $26 = cloud top pressure [hPa]\n"
00110         "# $27 = pressure at lifted condensation level (LCL) [hPa]\n"
00111         "# $28 = pressure at level of free convection (LFC) [hPa]\n"
00112         "# $29 = pressure at equilibrium level (EL) [hPa]\n"
00113         "# $30 = convective available potential energy (CAPE) [J/kg]\n");
00114 fprintf(out,
00115         "# $31 = relative humidity over water [%%]\n"
00116         "# $32 = relative humidity over ice [%%]\n"
00117         "# $33 = dew point temperature [K]\n"
00118         "# $34 = frost point temperature [K]\n");
00119
00120 /* Loop over air parcels... */
00121 for (ip = 0; ip < atm->np; ip++) {
00122
00123     /* Get meteorological data... */
00124     get_met(&ctl, atm->time[ip], &met0, &met1);
00125
00126     /* Set reference pressure for interpolation... */
00127     pref = atm->p[ip];
00128     if (geopot) {
00129         zref = Z(pref);
00130         p0 = met0->p[0];
00131         p1 = met0->p[met0->np - 1];
00132         for (it = 0; it < 24; it++) {
00133             pref = 0.5 * (p0 + p1);
00134             intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->time[ip], pref,
00135                               atm->lon[ip], atm->lat[ip], &zm, ci, cw, 1);
00136             if (zref > zm || !gsl_finite(zm))
00137                 p0 = pref;
00138             else
00139                 p1 = pref;
00140         }
00141         pref = 0.5 * (p0 + p1);
00142     }
00143
00144     /* Interpolate meteo data... */
00145     INTPOL_TIME_ALL(atm->time[ip], pref, atm->lon[ip], atm->lat[ip]);
00146
00147     /* Make blank lines... */
00148     if (ip == 0 || (grid_time && atm->time[ip] != time_old)
00149         || (grid_z && atm->p[ip] != p_old)
00150         || (grid_lon && atm->lon[ip] != lon_old)
00151         || (grid_lat && atm->lat[ip] != lat_old))
00152         fprintf(out, "\n");
00153     time_old = atm->time[ip];
00154     p_old = atm->p[ip];
00155     lon_old = atm->lon[ip];
00156     lat_old = atm->lat[ip];
00157
00158     /* Write data... */
00159     fprintf(out,
00160             "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00161             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
00162             atm->p[ip], t, u, v, w, h2o, o3, z, pv, ps, ts, zs, us, vs,
00163             pt, zt, tt, h2ot, lwc, iwc, cl, pc, plcl, plfc, pel, cape,
00164             RH(atm->p[ip], t, h2o), RHICE(atm->p[ip], t, h2o),
00165             TDEW(atm->p[ip], h2o), TICE(atm->p[ip], h2o));
00166 }
00167
00168 /* Close file... */
00169 fclose(out);
00170
00171 /* Free... */
00172 free(atm);
00173 free(met0);
00174 free(met1);
00175
00176 return EXIT_SUCCESS;
00177 }
00178

```

## 5.31 met\_spec.c File Reference

### Functions

- void [fft\\_help](#) (double \*fcReal, double \*fcImag, int n)
- int [main](#) (int argc, char \*argv[])

### 5.31.1 Detailed Description

Extract zonal mean from meteorological data.

Definition in file [met\\_spec.c](#).

### 5.31.2 Function Documentation

**5.31.2.1** `fft_help()` `void fft_help (`  
     `double * fcReal,`  
     `double * fcImag,`  
     `int n )`

Definition at line 143 of file [met\\_spec.c](#).

```
00146     {
00147
00148     gsl_fft_complex_wavetable *wavetable;
00149     gsl_fft_complex_workspace *workspace;
00150
00151     double data[2 * PMAX];
00152
00153     int i;
00154
00155     /* Check size... */
00156     if (n > PMAX)
00157         ERRMSG("Too many data points!");
00158
00159     /* Allocate... */
00160     wavetable = gsl_fft_complex_wavetable_alloc((size_t) n);
00161     workspace = gsl_fft_complex_workspace_alloc((size_t) n);
00162
00163     /* Set data (real, complex)... */
00164     for (i = 0; i < n; i++) {
00165         data[2 * i] = fcReal[i];
00166         data[2 * i + 1] = fcImag[i];
00167     }
00168
00169     /* Calculate FFT... */
00170     gsl_fft_complex_forward(data, 1, (size_t) n, wavetable, workspace);
00171
00172     /* Copy data... */
00173     for (i = 0; i < n; i++) {
00174         fcReal[i] = data[2 * i];
00175         fcImag[i] = data[2 * i + 1];
00176     }
00177
00178     /* Free... */
00179     gsl_fft_complex_wavetable_free(wavetable);
00180     gsl_fft_complex_workspace_free(workspace);
00181 }
```

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

Definition at line 47 of file [met\\_spec.c](#).

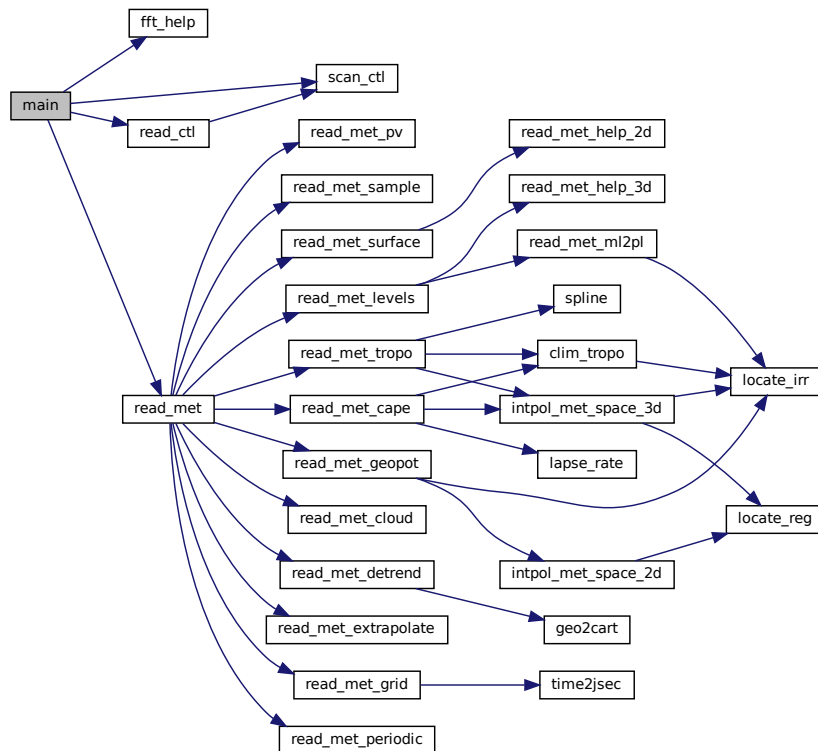
```
00049     {
00050
00051     ctl_t ctl;
00052
00053     met_t *met;
00054
00055     FILE *out;
00056
00057     static double cutImag[PMAX], cutReal[PMAX], lx[PMAX], A[PMAX], phi[PMAX],
```

```

00058     wavemax;
00059
00060     /* Allocate... */
00061     ALLOC(met, met_t, 1);
00062
00063     /* Check arguments... */
00064     if (argc < 4)
00065         ERRMSG("Give parameters: <ctl> <spec.tab> <met0>");
00066
00067     /* Read control parameters... */
00068     read_ctl(argv[1], argc, argv, &ctl);
00069     wavemax =
00070         (int) scan_ctl(argv[1], argc, argv, "SPEC_WAVEMAX", -1, "7", NULL);
00071
00072     /* Read meteorological data... */
00073     if (!read_met(&ctl, argv[3], met))
00074         ERRMSG("Cannot read meteo data!");
00075
00076     /* Create output file... */
00077     printf("Write spectral data file: %s\n", argv[2]);
00078     if (!(out = fopen(argv[2], "w")))
00079         ERRMSG("Cannot create file!");
00080
00081     /* Write header... */
00082     fprintf(out,
00083         "# $1 = time [s]\n"
00084         "# $2 = altitude [km]\n"
00085         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00086     for (int ix = 0; ix <= wavemax; ix++) {
00087         fprintf(out, "# $d = wavelength (PW%d) [km]\n", 5 + 3 * ix, ix);
00088         fprintf(out, "# $d = amplitude (PW%d) [K]\n", 6 + 3 * ix, ix);
00089         fprintf(out, "# $d = phase (PW%d) [deg]\n", 7 + 3 * ix, ix);
00090     }
00091
00092     /* Loop over pressure levels... */
00093     for (int ip = 0; ip < met->np; ip++) {
00094
00095         /* Write output... */
00096         fprintf(out, "\n");
00097
00098         /* Loop over latitudes... */
00099         for (int iy = 0; iy < met->ny; iy++) {
00100
00101             /* Copy data... */
00102             for (int ix = 0; ix < met->nx; ix++) {
00103                 cutReal[ix] = met->t[ix][iy][ip];
00104                 cutImag[ix] = 0.0;
00105             }
00106
00107             /* FFT... */
00108             fft_help(cutReal, cutImag, met->nx);
00109
00110             /*
00111              * Get wavelength, amplitude, and phase:
00112              *  $A(x) = A[0] + A[1] * \cos(2 \pi x / lx[1] + \phi[1]) + A[2] * \cos...$ 
00113              */
00114             for (int ix = 0; ix < met->nx; ix++) {
00115                 lx[ix] = DEG2DX(met->lon[met->nx - 1] - met->lon[0], met->lat[iy])
00116                     / ((ix < met->nx / 2) ? (double) ix : -(double) (met->nx - ix));
00117                 A[ix] = (ix == 0 ? 1.0 : 2.0) / (met->nx)
00118                     * sqrt(gsl_pow_2(cutReal[ix]) + gsl_pow_2(cutImag[ix]));
00119                 phi[ix]
00120                     = 180. / M_PI * atan2(cutImag[ix], cutReal[ix]);
00121             }
00122
00123             /* Write data... */
00124             fprintf(out, "%.2f %g %g %g", met->time, Z(met->p[ip]), 0.0,
00125                 met->lat[iy]);
00126             for (int ix = 0; ix <= wavemax; ix++)
00127                 fprintf(out, " %g %g %g", lx[ix], A[ix], phi[ix]);
00128             fprintf(out, "\n");
00129         }
00130     }
00131
00132     /* Close file... */
00133     fclose(out);
00134
00135     /* Free... */
00136     free(met);
00137
00138     return EXIT_SUCCESS;
00139 }

```

Here is the call graph for this function:



## 5.32 met\_spec.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
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028  Dimensions...
00029  ----- */
00030
00032 #define PMAX EX
00033
00034 /* -----
00035  Functions...
00036  ----- */
00037
00038 void fft_help(
00039     double *fcReal,
00040     double *fcImag,
00041     int n);
00042
00043 /* -----

```



```

00044     Main...
00045     ----- */
00046
00047 int main(
00048     int argc,
00049     char *argv[]) {
00050
00051     ctl_t ctl;
00052
00053     met_t *met;
00054
00055     FILE *out;
00056
00057     static double cutImag[PMAX], cutReal[PMAX], lx[PMAX], A[PMAX], phi[PMAX],
00058         wavemax;
00059
00060     /* Allocate... */
00061     ALLOC(met, met_t, 1);
00062
00063     /* Check arguments... */
00064     if (argc < 4)
00065         ERRMSG("Give parameters: <ctl> <spec.tab> <met0>");
00066
00067     /* Read control parameters... */
00068     read_ctl(argv[1], argc, argv, &ctl);
00069     wavemax =
00070         (int) scan_ctl(argv[1], argc, argv, "SPEC_WAVEMAX", -1, "7", NULL);
00071
00072     /* Read meteorological data... */
00073     if (!read_met(&ctl, argv[3], met))
00074         ERRMSG("Cannot read meteo data!");
00075
00076     /* Create output file... */
00077     printf("Write spectral data file: %s\n", argv[2]);
00078     if (!(out = fopen(argv[2], "w")))
00079         ERRMSG("Cannot create file!");
00080
00081     /* Write header... */
00082     fprintf(out,
00083         "# $1 = time [s]\n"
00084         "# $2 = altitude [km]\n"
00085         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00086     for (int ix = 0; ix <= wavemax; ix++) {
00087         fprintf(out, "# $d = wavelength (PW%d) [km]\n", 5 + 3 * ix, ix);
00088         fprintf(out, "# $d = amplitude (PW%d) [K]\n", 6 + 3 * ix, ix);
00089         fprintf(out, "# $d = phase (PW%d) [deg]\n", 7 + 3 * ix, ix);
00090     }
00091
00092     /* Loop over pressure levels... */
00093     for (int ip = 0; ip < met->np; ip++) {
00094
00095         /* Write output... */
00096         fprintf(out, "\n");
00097
00098         /* Loop over latitudes... */
00099         for (int iy = 0; iy < met->ny; iy++) {
00100
00101             /* Copy data... */
00102             for (int ix = 0; ix < met->nx; ix++) {
00103                 cutReal[ix] = met->t[ix][iy][ip];
00104                 cutImag[ix] = 0.0;
00105             }
00106
00107             /* FFT... */
00108             fft_help(cutReal, cutImag, met->nx);
00109
00110             /*
00111              * Get wavelength, amplitude, and phase:
00112              * A(x) = A[0] + A[1] * cos(2 pi x / lx[1] + phi[1]) + A[2] * cos...
00113              */
00114             for (int ix = 0; ix < met->nx; ix++) {
00115                 lx[ix] = DEG2DX(met->lon[met->nx - 1] - met->lon[0], met->lat[iy])
00116                     / ((ix < met->nx / 2) ? (double) ix : -(double) (met->nx - ix));
00117                 A[ix] = (ix == 0 ? 1.0 : 2.0) / (met->nx)
00118                     * sqrt(gsl_pow_2(cutReal[ix]) + gsl_pow_2(cutImag[ix]));
00119                 phi[ix]
00120                     = 180. / M_PI * atan2(cutImag[ix], cutReal[ix]);
00121             }
00122
00123             /* Write data... */
00124             fprintf(out, "%.2f %g %g %g", met->time, Z(met->p[ip]), 0.0,
00125                 met->lat[iy]);
00126             for (int ix = 0; ix <= wavemax; ix++)
00127                 fprintf(out, " %g %g %g", lx[ix], A[ix], phi[ix]);
00128             fprintf(out, "\n");
00129         }
00130     }

```

```

00131
00132  /* Close file... */
00133  fclose(out);
00134
00135  /* Free... */
00136  free(met);
00137
00138  return EXIT_SUCCESS;
00139 }
00140
00141 /*****
00142
00143 void fft_help(
00144     double *fcReal,
00145     double *fcImag,
00146     int n) {
00147
00148     gsl_fft_complex_wavetable *wavetable;
00149     gsl_fft_complex_workspace *workspace;
00150
00151     double data[2 * PMAX];
00152
00153     int i;
00154
00155     /* Check size... */
00156     if (n > PMAX)
00157         ERRMSG("Too many data points!");
00158
00159     /* Allocate... */
00160     wavetable = gsl_fft_complex_wavetable_alloc((size_t) n);
00161     workspace = gsl_fft_complex_workspace_alloc((size_t) n);
00162
00163     /* Set data (real, complex)... */
00164     for (i = 0; i < n; i++) {
00165         data[2 * i] = fcReal[i];
00166         data[2 * i + 1] = fcImag[i];
00167     }
00168
00169     /* Calculate FFT... */
00170     gsl_fft_complex_forward(data, 1, (size_t) n, wavetable, workspace);
00171
00172     /* Copy data... */
00173     for (i = 0; i < n; i++) {
00174         fcReal[i] = data[2 * i];
00175         fcImag[i] = data[2 * i + 1];
00176     }
00177
00178     /* Free... */
00179     gsl_fft_complex_wavetable_free(wavetable);
00180     gsl_fft_complex_workspace_free(workspace);
00181 }

```

## 5.33 met\_zm.c File Reference

### Functions

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

#### 5.33.1 Detailed Description

Extract zonal mean from meteorological data.

Definition in file [met\\_zm.c](#).

#### 5.33.2 Function Documentation

```

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

```

Definition at line 41 of file [met\\_zm.c](#).

```

00043     {
00044
00045         ctl_t ctl;
00046
00047         met_t *met;
00048
00049         FILE *out;
00050
00051         static double timem[NZ][NY], psm[NZ][NY], tsm[NZ][NY], zsm[NZ][NY],
00052             usm[NZ][NY], vsm[NZ][NY], ptm[NZ][NY], pcm[NZ][NY],
00053             clm[NZ][NY], plclm[NZ][NY], plfcm[NZ][NY], pelm[NZ][NY], capem[NZ][NY],
00054             ttm[NZ][NY], ztm[NZ][NY], tm[NZ][NY], um[NZ][NY], vm[NZ][NY],
00055             wm[NZ][NY], h2om[NZ][NY], h2otm[NZ][NY], pvm[NZ][NY], o3m[NZ][NY],
00056             lwc[NZ][NY], iwc[NZ][NY], zm[NZ][NY], z, z0, z1, dz, zt, tt, plev[NZ],
00057             ps, ts, zs, us, vs, pt, pc, plcl, plfc, pel, cape, cl, t, u, v, w, pv,
00058             h2o, h2ot, o3, lwc, iwc, lat, lat0, lat1, dlat, lats[NY], cw[3];
00059
00060         static int i, ix, iy, iz, np[NZ][NY], npt[NZ][NY], ny, nz, ci[3];
00061
00062         /* Allocate... */
00063         ALLOC(met, met_t, 1);
00064
00065         /* Check arguments... */
00066         if (argc < 4)
00067             ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00068
00069         /* Read control parameters... */
00070         read_ctl(argv[1], argc, argv, &ctl);
00071         z0 = scan_ctl(argv[1], argc, argv, "ZM_Z0", -1, "-999", NULL);
00072         z1 = scan_ctl(argv[1], argc, argv, "ZM_Z1", -1, "-999", NULL);
00073         dz = scan_ctl(argv[1], argc, argv, "ZM_DZ", -1, "-999", NULL);
00074         lat0 = scan_ctl(argv[1], argc, argv, "ZM_LAT0", -1, "-90", NULL);
00075         lat1 = scan_ctl(argv[1], argc, argv, "ZM_LAT1", -1, "90", NULL);
00076         dlat = scan_ctl(argv[1], argc, argv, "ZM_DLAT", -1, "-999", NULL);
00077
00078         /* Loop over files... */
00079         for (i = 3; i < argc; i++) {
00080
00081             /* Read meteorological data... */
00082             if (!read_met(&ctl, argv[i], met))
00083                 continue;
00084
00085             /* Set vertical grid... */
00086             if (z0 < 0)
00087                 z0 = Z(met->p[0]);
00088             if (z1 < 0)
00089                 z1 = Z(met->p[met->np - 1]);
00090             nz = 0;
00091             if (dz < 0) {
00092                 for (iz = 0; iz < met->np; iz++)
00093                     if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00094                         plev[nz] = met->p[iz];
00095                         if ((++nz) > NZ)
00096                             ERRMSG("Too many pressure levels!");
00097                     }
00098             } else
00099                 for (z = z0; z <= z1; z += dz) {
00100                     plev[nz] = P(z);
00101                     if ((++nz) > NZ)
00102                         ERRMSG("Too many pressure levels!");
00103                 }
00104
00105             /* Set horizontal grid... */
00106             if (dlat <= 0)
00107                 dlat = fabs(met->lat[1] - met->lat[0]);
00108             ny = 0;
00109             if (lat0 < -90 && lat1 > 90) {
00110                 lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00111                 lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00112             }
00113             for (lat = lat0; lat <= lat1; lat += dlat) {
00114                 lats[ny] = lat;
00115                 if ((++ny) > NY)
00116                     ERRMSG("Too many latitudes!");
00117             }
00118
00119             /* Average... */
00120             for (ix = 0; ix < met->nx; ix++)
00121                 for (iy = 0; iy < ny; iy++)
00122                     for (iz = 0; iz < nz; iz++) {

```

```

00123
00124     /* Interpolate meteo data... */
00125     INTPOL_SPACE_ALL(plev[iz], met->lon[ix], met->lat[iy]);
00126
00127     /* Averaging... */
00128     timem[iz][iy] += met->time;
00129     zm[iz][iy] += z;
00130     tm[iz][iy] += t;
00131     um[iz][iy] += u;
00132     vm[iz][iy] += v;
00133     wm[iz][iy] += w;
00134     pvm[iz][iy] += pv;
00135     h2om[iz][iy] += h2o;
00136     o3m[iz][iy] += o3;
00137     lwcm[iz][iy] += lwc;
00138     iwcm[iz][iy] += iwc;
00139     psm[iz][iy] += ps;
00140     tsm[iz][iy] += ts;
00141     zsm[iz][iy] += zs;
00142     usm[iz][iy] += us;
00143     vsm[iz][iy] += vs;
00144     pcm[iz][iy] += pc;
00145     clm[iz][iy] += cl;
00146     plclm[iz][iy] += plcl;
00147     plfcm[iz][iy] += plfc;
00148     pelm[iz][iy] += pel;
00149     capem[iz][iy] += cape;
00150     if (gsl_finite(pt)) {
00151         ptm[iz][iy] += pt;
00152         ztm[iz][iy] += zt;
00153         ttm[iz][iy] += tt;
00154         h2otm[iz][iy] += h2ot;
00155         npt[iz][iy]++;
00156     }
00157     np[iz][iy]++;
00158 }
00159 }
00160
00161 /* Create output file... */
00162 printf("Write meteorological data file: %s\n", argv[2]);
00163 if (!(out = fopen(argv[2], "w")))
00164     ERRMSG("Cannot create file!");
00165
00166 /* Write header... */
00167 fprintf(out,
00168     "# $1 = time [s]\n"
00169     "# $2 = altitude [km]\n"
00170     "# $3 = longitude [deg]\n"
00171     "# $4 = latitude [deg]\n"
00172     "# $5 = pressure [hPa]\n"
00173     "# $6 = temperature [K]\n"
00174     "# $7 = zonal wind [m/s]\n"
00175     "# $8 = meridional wind [m/s]\n"
00176     "# $9 = vertical velocity [hPa/s]\n"
00177     "# $10 = H2O volume mixing ratio [ppv]\n");
00178 fprintf(out,
00179     "# $11 = O3 volume mixing ratio [ppv]\n"
00180     "# $12 = geopotential height [km]\n"
00181     "# $13 = potential vorticity [PVU]\n"
00182     "# $14 = surface pressure [hPa]\n"
00183     "# $15 = surface temperature [K]\n"
00184     "# $16 = surface geopotential height [km]\n"
00185     "# $17 = surface zonal wind [m/s]\n"
00186     "# $18 = surface meridional wind [m/s]\n"
00187     "# $19 = tropopause pressure [hPa]\n"
00188     "# $20 = tropopause geopotential height [km]\n");
00189 fprintf(out,
00190     "# $21 = tropopause temperature [K]\n"
00191     "# $22 = tropopause water vapor [ppv]\n"
00192     "# $23 = cloud liquid water content [kg/kg]\n"
00193     "# $24 = cloud ice water content [kg/kg]\n"
00194     "# $25 = total column cloud water [kg/m^2]\n"
00195     "# $26 = cloud top pressure [hPa]\n"
00196     "# $27 = pressure at lifted condensation level (LCL) [hPa]\n"
00197     "# $28 = pressure at level of free convection (LFC) [hPa]\n"
00198     "# $29 = pressure at equilibrium level (EL) [hPa]\n"
00199     "# $30 = convective available potential energy (CAPE) [J/kg]\n");
00200 fprintf(out,
00201     "# $31 = relative humidity over water [%]\n"
00202     "# $32 = relative humidity over ice [%]\n"
00203     "# $33 = dew point temperature [K]\n"
00204     "# $34 = frost point temperature [K]\n");
00205
00206 /* Write data... */
00207 for (iz = 0; iz < nz; iz++) {
00208     fprintf(out, "\n");
00209     for (iy = 0; iy < ny; iy++)

```



```

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
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018  */
00019
00025  #include "libtrac.h"
00026
00027  /* -----
00028      Dimensions...
00029      ----- */
00030
00032  #define NZ 1000
00033
00035  #define NY 721
00036
00037  /* -----
00038      Main...
00039      ----- */
00040
00041  int main(
00042      int argc,
00043      char *argv[]) {
00044
00045      ctl_t ctl;
00046
00047      met_t *met;
00048
00049      FILE *out;
00050
00051      static double timem[NZ][NY], psm[NZ][NY], tsm[NZ][NY], zsm[NZ][NY],
00052          usm[NZ][NY], vsm[NZ][NY], ptm[NZ][NY], pcm[NZ][NY],
00053          clm[NZ][NY], plclm[NZ][NY], plfcm[NZ][NY], pelm[NZ][NY], capem[NZ][NY],
00054          ttm[NZ][NY], ztm[NZ][NY], tm[NZ][NY], um[NZ][NY], vm[NZ][NY],
00055          wm[NZ][NY], h2om[NZ][NY], h2otm[NZ][NY], pvm[NZ][NY], o3m[NZ][NY],
00056          lwc[NZ][NY], iwc[NZ][NY], zm[NZ][NY], z, z0, z1, dz, zt, tt, plev[NZ],
00057          ps, ts, zs, us, vs, pt, pc, plcl, plfc, pel, cape, cl, t, u, v, w, pv,
00058          h2o, h2ot, o3, lwc, iwc, lat, lat0, lat1, dlat, lats[NY], cw[3];
00059
00060      static int i, ix, iy, iz, np[NZ][NY], npt[NZ][NY], ny, nz, ci[3];
00061
00062      /* Allocate... */
00063      ALLOC(met, met_t, 1);
00064
00065      /* Check arguments... */
00066      if (argc < 4)
00067          ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00068
00069      /* Read control parameters... */
00070      read_ctl(argv[1], argc, argv, &ctl);
00071      z0 = scan_ctl(argv[1], argc, argv, "ZM_Z0", -1, "-999", NULL);
00072      z1 = scan_ctl(argv[1], argc, argv, "ZM_Z1", -1, "-999", NULL);
00073      dz = scan_ctl(argv[1], argc, argv, "ZM_DZ", -1, "-999", NULL);
00074      lat0 = scan_ctl(argv[1], argc, argv, "ZM_LAT0", -1, "-90", NULL);
00075      lat1 = scan_ctl(argv[1], argc, argv, "ZM_LAT1", -1, "90", NULL);
00076      dlat = scan_ctl(argv[1], argc, argv, "ZM_DLAT", -1, "-999", NULL);
00077
00078      /* Loop over files... */
00079      for (i = 3; i < argc; i++) {
00080
00081          /* Read meteorological data... */
00082          if (!read_met(&ctl, argv[i], met))
00083              continue;
00084
00085          /* Set vertical grid... */
00086          if (z0 < 0)
00087              z0 = Z(met->p[0]);
00088          if (z1 < 0)
00089              z1 = Z(met->p[met->np - 1]);
00090          nz = 0;
00091          if (dz < 0) {
00092              for (iz = 0; iz < met->np; iz++)
00093                  if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00094                      plev[nz] = met->p[iz];
00095                      if ((++nz) > NZ)
00096                          ERRMSG("Too many pressure levels!");
00097                  }
00098              } else

```

```

00099     for (z = z0; z <= z1; z += dz) {
00100         plev[nz] = P(z);
00101         if ((++nz) > NZ)
00102             ERRMSG("Too many pressure levels!");
00103     }
00104
00105     /* Set horizontal grid... */
00106     if (dlat <= 0)
00107         dlat = fabs(met->lat[1] - met->lat[0]);
00108     ny = 0;
00109     if (lat0 < -90 && lat1 > 90) {
00110         lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00111         lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00112     }
00113     for (lat = lat0; lat <= lat1; lat += dlat) {
00114         lats[ny] = lat;
00115         if ((++ny) > NY)
00116             ERRMSG("Too many latitudes!");
00117     }
00118
00119     /* Average... */
00120     for (ix = 0; ix < met->nx; ix++)
00121         for (iy = 0; iy < ny; iy++)
00122             for (iz = 0; iz < nz; iz++) {
00123
00124                 /* Interpolate meteo data... */
00125                 INTPOL_SPACE_ALL(plev[iz], met->lon[ix], met->lat[iy]);
00126
00127                 /* Averaging... */
00128                 timem[iz][iy] += met->time;
00129                 zm[iz][iy] += z;
00130                 tm[iz][iy] += t;
00131                 um[iz][iy] += u;
00132                 vm[iz][iy] += v;
00133                 wm[iz][iy] += w;
00134                 pvm[iz][iy] += pv;
00135                 h2om[iz][iy] += h2o;
00136                 o3m[iz][iy] += o3;
00137                 lwcm[iz][iy] += lwc;
00138                 iwcm[iz][iy] += iwc;
00139                 psm[iz][iy] += ps;
00140                 tsm[iz][iy] += ts;
00141                 zsm[iz][iy] += zs;
00142                 usm[iz][iy] += us;
00143                 vsm[iz][iy] += vs;
00144                 pcm[iz][iy] += pc;
00145                 clm[iz][iy] += cl;
00146                 plclm[iz][iy] += plcl;
00147                 plfcm[iz][iy] += plfc;
00148                 pelm[iz][iy] += pel;
00149                 capem[iz][iy] += cape;
00150                 if (gsl_finite(pt)) {
00151                     ptm[iz][iy] += pt;
00152                     ztm[iz][iy] += zt;
00153                     ttm[iz][iy] += tt;
00154                     h2otm[iz][iy] += h2ot;
00155                     npt[iz][iy]++;
00156                 }
00157                 np[iz][iy]++;
00158             }
00159 }
00160
00161 /* Create output file... */
00162 printf("Write meteorological data file: %s\n", argv[2]);
00163 if (!out = fopen(argv[2], "w"))
00164     ERRMSG("Cannot create file!");
00165
00166 /* Write header... */
00167 fprintf(out,
00168     "# $1 = time [s]\n"
00169     "# $2 = altitude [km]\n"
00170     "# $3 = longitude [deg]\n"
00171     "# $4 = latitude [deg]\n"
00172     "# $5 = pressure [hPa]\n"
00173     "# $6 = temperature [K]\n"
00174     "# $7 = zonal wind [m/s]\n"
00175     "# $8 = meridional wind [m/s]\n"
00176     "# $9 = vertical velocity [hPa/s]\n"
00177     "# $10 = H2O volume mixing ratio [ppv]\n");
00178 fprintf(out,
00179     "# $11 = O3 volume mixing ratio [ppv]\n"
00180     "# $12 = geopotential height [km]\n"
00181     "# $13 = potential vorticity [PVU]\n"
00182     "# $14 = surface pressure [hPa]\n"
00183     "# $15 = surface temperature [K]\n"
00184     "# $16 = surface geopotential height [km]\n"
00185     "# $17 = surface zonal wind [m/s]\n"

```

```
00186     "# $8 = surface meridional wind [m/s]\n"
00187     "# $19 = tropopause pressure [hPa]\n"
00188     "# $20 = tropopause geopotential height [km]\n");
00189 fprintf(out,
00190     "# $21 = tropopause temperature [K]\n"
00191     "# $22 = tropopause water vapor [ppv]\n"
00192     "# $23 = cloud liquid water content [kg/kg]\n"
00193     "# $24 = cloud ice water content [kg/kg]\n"
00194     "# $25 = total column cloud water [kg/m^2]\n"
00195     "# $26 = cloud top pressure [hPa]\n"
00196     "# $27 = pressure at lifted condensation level (LCL) [hPa]\n"
00197     "# $28 = pressure at level of free convection (LFC) [hPa]\n"
00198     "# $29 = pressure at equilibrium level (EL) [hPa]\n"
00199     "# $30 = convective available potential energy (CAPE) [J/kg]\n");
00200 fprintf(out,
00201     "# $31 = relative humidity over water [%]\n"
00202     "# $32 = relative humidity over ice [%]\n"
00203     "# $33 = dew point temperature [K]\n"
00204     "# $34 = frost point temperature [K]\n");
00205
00206 /* Write data... */
00207 for (iz = 0; iz < nz; iz++) {
00208     fprintf(out, "\n");
00209     for (iy = 0; iy < ny; iy++)
00210         fprintf(out,
00211             "%2.f %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00212             " %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00213             timem[iz][iy] / np[iz][iy], Z(plev[iz]), 0.0, lats[iy],
00214             plev[iz], tm[iz][iy] / np[iz][iy], um[iz][iy] / np[iz][iy],
00215             vm[iz][iy] / np[iz][iy], wm[iz][iy] / np[iz][iy],
00216             h2om[iz][iy] / np[iz][iy], o3m[iz][iy] / np[iz][iy],
00217             zm[iz][iy] / np[iz][iy], pvmm[iz][iy] / np[iz][iy],
00218             psm[iz][iy] / np[iz][iy], tsm[iz][iy] / np[iz][iy],
00219             zsm[iz][iy] / np[iz][iy], usm[iz][iy] / np[iz][iy],
00220             vsm[iz][iy] / np[iz][iy], ptm[iz][iy] / npt[iz][iy],
00221             ztm[iz][iy] / npt[iz][iy], ttmm[iz][iy] / npt[iz][iy],
00222             h2otm[iz][iy] / npt[iz][iy], lwcm[iz][iy] / np[iz][iy],
00223             iwcm[iz][iy] / np[iz][iy], clm[iz][iy] / np[iz][iy],
00224             pcm[iz][iy] / np[iz][iy], plclm[iz][iy] / np[iz][iy],
00225             plfcm[iz][iy] / np[iz][iy], pelm[iz][iy] / np[iz][iy],
00226             capem[iz][iy] / np[iz][iy],
00227             RH(plev[iz], tm[iz][iy] / np[iz][iy],
00228                 h2om[iz][iy] / np[iz][iy]),
00229             RHICE(plev[iz], tm[iz][iy] / np[iz][iy],
00230                 h2om[iz][iy] / np[iz][iy]),
00231             TDEW(plev[iz], h2om[iz][iy] / np[iz][iy]),
00232             TICE(plev[iz], h2om[iz][iy] / np[iz][iy]));
00233 }
00234
00235 /* Close file... */
00236 fclose(out);
00237
00238 /* Free... */
00239 free(met);
00240
00241 return EXIT_SUCCESS;
00242 }
```

### 5.35 time2jsec.c File Reference

## Functions

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

### 5.35.1 Detailed Description

Convert date to Julian seconds.

Definition in file [time2jsec.c](#).

### 5.35.2 Function Documentation



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

Here is the call graph for this function:



## 5.36 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.
00008
00009  MPTRAC is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU General Public License for more details.
00013
00014  You should have received a copy of the GNU General Public License
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00020 #include "libtrac.h"
00021
00022 int main(
00023     int argc,
00024     char *argv[]) {
00025
00026     double jsec, remain;
00027
00028     int day, hour, min, mon, sec, year;
00029
00030     /* Check arguments... */
00031     if (argc < 8)
```

```

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

```

## 5.37 trac.c File Reference

### Functions

- void `module_advection` (`met_t` \*met0, `met_t` \*met1, `atm_t` \*atm, double \*dt)  
*Calculate advection of air parcels.*
- void `module_convection` (`ctl_t` \*ctl, `met_t` \*met0, `met_t` \*met1, `atm_t` \*atm, double \*dt, double \*rs)  
*Calculate convection of air parcels.*
- void `module_decay` (`ctl_t` \*ctl, `atm_t` \*atm, double \*dt)  
*Calculate exponential decay of particle mass.*
- void `module_diffusion_meso` (`ctl_t` \*ctl, `met_t` \*met0, `met_t` \*met1, `atm_t` \*atm, `cache_t` \*cache, double \*dt, double \*rs)  
*Calculate mesoscale diffusion.*
- void `module_diffusion_turb` (`ctl_t` \*ctl, `atm_t` \*atm, double \*dt, double \*rs)  
*Calculate turbulent diffusion.*
- void `module_dry_deposition` (`ctl_t` \*ctl, `met_t` \*met0, `met_t` \*met1, `atm_t` \*atm, double \*dt)  
*Calculate dry deposition.*
- void `module_isosurf_init` (`ctl_t` \*ctl, `met_t` \*met0, `met_t` \*met1, `atm_t` \*atm, `cache_t` \*cache)  
*Initialize isosurface module.*
- void `module_isosurf` (`ctl_t` \*ctl, `met_t` \*met0, `met_t` \*met1, `atm_t` \*atm, `cache_t` \*cache)  
*Force air parcels to stay on isosurface.*
- void `module_meteo` (`ctl_t` \*ctl, `met_t` \*met0, `met_t` \*met1, `atm_t` \*atm)  
*Interpolate meteorological data for air parcel positions.*
- void `module_position` (`met_t` \*met0, `met_t` \*met1, `atm_t` \*atm, double \*dt)  
*Check position of air parcels.*
- void `module_rng_init` (void)  
*Initialize random number generator...*
- void `module_rng` (double \*rs, size\_t n, int method)  
*Generate random numbers.*
- void `module_sedi` (`ctl_t` \*ctl, `met_t` \*met0, `met_t` \*met1, `atm_t` \*atm, double \*dt)  
*Calculate sedimentation of air parcels.*
- void `module_oh_chem` (`ctl_t` \*ctl, `met_t` \*met0, `met_t` \*met1, `atm_t` \*atm, double \*dt)  
*Calculate OH chemistry.*
- void `module_wet_deposition` (`ctl_t` \*ctl, `met_t` \*met0, `met_t` \*met1, `atm_t` \*atm, double \*dt)  
*Calculate wet deposition.*
- 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[])

## Variables

- `curandGenerator_t rng`

### 5.37.1 Detailed Description

Lagrangian particle dispersion model.

Definition in file [trac.c](#).

### 5.37.2 Function Documentation

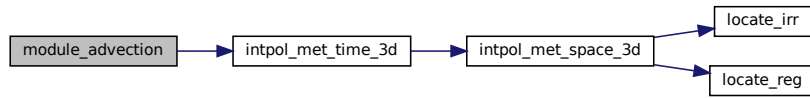
**5.37.2.1 module\_advection()** `void module_advection (`  
`met_t * met0,`  
`met_t * met1,`  
`atm_t * atm,`  
`double * dt )`

Calculate advection of air parcels.

Definition at line 450 of file [trac.c](#).

```
00454     {
00455
00456     /* Set timer... */
00457     SELECT_TIMER("MODULE_ADVECTION", NVTX_GPU);
00458
00459 #ifdef _OPENACC
00460 #pragma acc data present(met0,met1,atm,dt)
00461 #pragma acc parallel loop independent gang vector
00462 #else
00463 #pragma omp parallel for default(shared)
00464 #endif
00465     for (int ip = 0; ip < atm->np; ip++)
00466         if (dt[ip] != 0) {
00467
00468             double u, v, w;
00469
00470             /* Interpolate meteorological data... */
00471             INTPOL_INIT;
00472             INTPOL_3D(u, 1);
00473             INTPOL_3D(v, 0);
00474             INTPOL_3D(w, 0);
00475
00476             /* Get position of the mid point... */
00477             double dtm = atm->time[ip] + 0.5 * dt[ip];
00478             double xm0 =
00479                 atm->lon[ip] + DX2DEG(0.5 * dt[ip] * u / 1000., atm->lat[ip]);
00480             double xm1 = atm->lat[ip] + DY2DEG(0.5 * dt[ip] * v / 1000.);
00481             double xm2 = atm->p[ip] + 0.5 * dt[ip] * w;
00482
00483             /* Interpolate meteorological data for mid point... */
00484             intpol_met_time_3d(met0, met0->u, met1, met1->u,
00485                               dtm, xm2, xm0, xm1, &u, ci, cw, 1);
00486             intpol_met_time_3d(met0, met0->v, met1, met1->v,
00487                               dtm, xm2, xm0, xm1, &v, ci, cw, 0);
00488             intpol_met_time_3d(met0, met0->w, met1, met1->w,
00489                               dtm, xm2, xm0, xm1, &w, ci, cw, 0);
00490
00491             /* Save new position... */
00492             atm->time[ip] += dt[ip];
00493             atm->lon[ip] += DX2DEG(dt[ip] * u / 1000., xm1);
00494             atm->lat[ip] += DY2DEG(dt[ip] * v / 1000.);
00495             atm->p[ip] += dt[ip] * w;
00496         }
00497 }
```

Here is the call graph for this function:



**5.37.2.2 module\_convection()** void module\_convection (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt,
    double * rs )

```

Calculate convection of air parcels.

Definition at line 501 of file trac.c.

```

00507     {
00508
00509     /* Set timer... */
00510     SELECT_TIMER("MODULE_CONVECTION", NVTX_GPU);
00511
00512     #ifdef _OPENACC
00513     #pragma acc data present(ctl,met0,met1,atm,dt,rs)
00514     #pragma acc parallel loop independent gang vector
00515     #else
00516     #pragma omp parallel for default(shared)
00517     #endif
00518     for (int ip = 0; ip < atm->np; ip++)
00519     if (dt[ip] != 0) {
00520
00521         double cape, pel, ps;
00522
00523         /* Interpolate CAPE... */
00524         INTPOL_INIT;
00525         INTPOL_2D(cape, 1);
00526
00527         /* Check threshold... */
00528         if (isfinite(cape) && cape >= ctl->conv_cape) {
00529
00530             /* Interpolate equilibrium level... */
00531             INTPOL_2D(pel, 0);
00532
00533             /* Check whether particle is above cloud top... */
00534             if (!isfinite(pel) || atm->p[ip] < pel)
00535                 continue;
00536
00537             /* Interpolate surface pressure... */
00538             INTPOL_2D(ps, 0);
00539
00540             /* Redistribute particle in cloud column... */
00541             atm->p[ip] = ps + (pel - ps) * rs[ip];
00542         }
00543     }
00544 }

```

**5.37.2.3 module\_decay()** void module\_decay (

```

    ctl_t * ctl,
    atm_t * atm,
    double * dt )

```

Calculate exponential decay of particle mass.

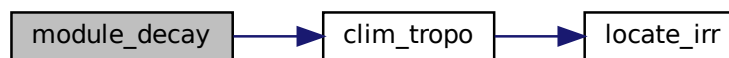
Definition at line 548 of file trac.c.

```

00551     {
00552
00553     /* Set timer... */
00554     SELECT_TIMER("MODULE_DECAY", NVTX_GPU);
00555
00556     /* Check quantity flags... */
00557     if (ctl->qnt_m < 0)
00558         ERRMSG("Module needs quantity mass!");
00559
00560 #ifdef _OPENACC
00561 #pragma acc data present(ctl,atm,dt)
00562 #pragma acc parallel loop independent gang vector
00563 #else
00564 #pragma omp parallel for default(shared)
00565 #endif
00566     for (int ip = 0; ip < atm->np; ip++)
00567         if (dt[ip] != 0) {
00568
00569             /* Get tropopause pressure... */
00570             double pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00571
00572             /* Get weighting factor... */
00573             double w;
00574             double p1 = pt * 0.866877899;
00575             double p0 = pt / 0.866877899;
00576             if (atm->p[ip] > p0)
00577                 w = 1;
00578             else if (atm->p[ip] < p1)
00579                 w = 0;
00580             else
00581                 w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00582
00583             /* Set lifetime... */
00584             double tdec = w * ctl->tdec_trop + (1 - w) * ctl->tdec_strat;
00585
00586             /* Calculate exponential decay... */
00587             atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] / tdec);
00588         }
00589 }

```

Here is the call graph for this function:



**5.37.2.4 module\_diffusion\_meso()** void module\_diffusion\_meso (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    cache_t * cache,
    double * dt,
    double * rs )

```

Calculate mesoscale diffusion.

Definition at line 593 of file trac.c.

```

00600     {
00601
00602     /* Set timer... */
00603     SELECT_TIMER("MODULE_TURBMESO", NVTX_GPU);
00604
00605 #ifndef _OPENACC
00606 #pragma acc data present(ctl,met0,met1,atm,cache,dt,rs)
00607 #pragma acc parallel loop independent gang vector
00608 #else
00609 #pragma omp parallel for default(shared)
00610 #endif
00611     for (int ip = 0; ip < atm->np; ip++)
00612         if (dt[ip] != 0) {
00613
00614             double u[16], v[16], w[16];
00615
00616             /* Get indices... */
00617             int ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
00618             int iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
00619             int iz = locate_irr(met0->p, met0->np, atm->p[ip]);
00620
00621             /* Caching of wind standard deviations... */
00622             if (cache->tsig[ix][iy][iz] != met0->time) {
00623
00624                 /* Collect local wind data... */
00625                 u[0] = met0->u[ix][iy][iz];
00626                 u[1] = met0->u[ix + 1][iy][iz];
00627                 u[2] = met0->u[ix][iy + 1][iz];
00628                 u[3] = met0->u[ix + 1][iy + 1][iz];
00629                 u[4] = met0->u[ix][iy][iz + 1];
00630                 u[5] = met0->u[ix + 1][iy][iz + 1];
00631                 u[6] = met0->u[ix][iy + 1][iz + 1];
00632                 u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00633
00634                 v[0] = met0->v[ix][iy][iz];
00635                 v[1] = met0->v[ix + 1][iy][iz];
00636                 v[2] = met0->v[ix][iy + 1][iz];
00637                 v[3] = met0->v[ix + 1][iy + 1][iz];
00638                 v[4] = met0->v[ix][iy][iz + 1];
00639                 v[5] = met0->v[ix + 1][iy][iz + 1];
00640                 v[6] = met0->v[ix][iy + 1][iz + 1];
00641                 v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00642
00643                 w[0] = met0->w[ix][iy][iz];
00644                 w[1] = met0->w[ix + 1][iy][iz];
00645                 w[2] = met0->w[ix][iy + 1][iz];
00646                 w[3] = met0->w[ix + 1][iy + 1][iz];
00647                 w[4] = met0->w[ix][iy][iz + 1];
00648                 w[5] = met0->w[ix + 1][iy][iz + 1];
00649                 w[6] = met0->w[ix][iy + 1][iz + 1];
00650                 w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00651
00652                 /* Collect local wind data... */
00653                 u[8] = met1->u[ix][iy][iz];
00654                 u[9] = met1->u[ix + 1][iy][iz];
00655                 u[10] = met1->u[ix][iy + 1][iz];
00656                 u[11] = met1->u[ix + 1][iy + 1][iz];
00657                 u[12] = met1->u[ix][iy][iz + 1];
00658                 u[13] = met1->u[ix + 1][iy][iz + 1];
00659                 u[14] = met1->u[ix][iy + 1][iz + 1];
00660                 u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00661
00662                 v[8] = met1->v[ix][iy][iz];
00663                 v[9] = met1->v[ix + 1][iy][iz];
00664                 v[10] = met1->v[ix][iy + 1][iz];
00665                 v[11] = met1->v[ix + 1][iy + 1][iz];
00666                 v[12] = met1->v[ix][iy][iz + 1];
00667                 v[13] = met1->v[ix + 1][iy][iz + 1];
00668                 v[14] = met1->v[ix][iy + 1][iz + 1];
00669                 v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00670
00671                 w[8] = met1->w[ix][iy][iz];
00672                 w[9] = met1->w[ix + 1][iy][iz];
00673                 w[10] = met1->w[ix][iy + 1][iz];
00674                 w[11] = met1->w[ix + 1][iy + 1][iz];
00675                 w[12] = met1->w[ix][iy][iz + 1];
00676                 w[13] = met1->w[ix + 1][iy][iz + 1];
00677                 w[14] = met1->w[ix][iy + 1][iz + 1];
00678                 w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00679
00680                 /* Get standard deviations of local wind data... */
00681                 cache->usig[ix][iy][iz] = (float) stddev(u, 16);
00682                 cache->vsig[ix][iy][iz] = (float) stddev(v, 16);

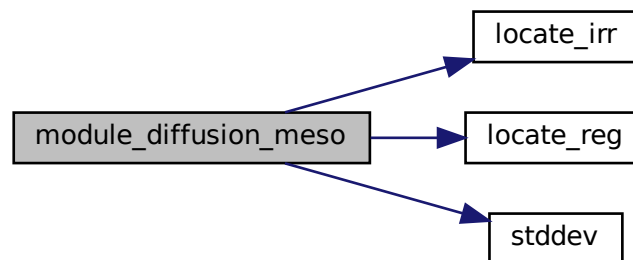
```

```

00683     cache->wsig[ix][iy][iz] = (float) stddev(w, 16);
00684     cache->tsig[ix][iy][iz] = met0->time;
00685 }
00686
00687 /* Set temporal correlations for mesoscale fluctuations... */
00688 double r = 1 - 2 * fabs(dt[ip]) / ctl->dt_met;
00689 double r2 = sqrt(1 - r * r);
00690
00691 /* Calculate horizontal mesoscale wind fluctuations... */
00692 if (ctl->turb_mesox > 0) {
00693     cache->up[ip] = (float)
00694         (r * cache->up[ip]
00695          + r2 * rs[3 * ip] * ctl->turb_mesox * cache->usig[ix][iy][iz]);
00696     atm->lon[ip] += DX2DEG(cache->up[ip] * dt[ip] / 1000., atm->lat[ip]);
00697
00698     cache->vp[ip] = (float)
00699         (r * cache->vp[ip]
00700          + r2 * rs[3 * ip + 1] * ctl->turb_mesox * cache->vsig[ix][iy][iz]);
00701     atm->lat[ip] += DY2DEG(cache->vp[ip] * dt[ip] / 1000.);
00702 }
00703
00704 /* Calculate vertical mesoscale wind fluctuations... */
00705 if (ctl->turb_mesoz > 0) {
00706     cache->wp[ip] = (float)
00707         (r * cache->wp[ip]
00708          + r2 * rs[3 * ip + 2] * ctl->turb_mesoz * cache->wsig[ix][iy][iz]);
00709     atm->p[ip] += cache->wp[ip] * dt[ip];
00710 }
00711 }
00712 }

```

Here is the call graph for this function:



**5.37.2.5 module\_diffusion\_turb()** void module\_diffusion\_turb (

```

    ctl_t * ctl,
    atm_t * atm,
    double * dt,
    double * rs )

```

Calculate turbulent diffusion.

Definition at line 716 of file [trac.c](#).

```

00720     {
00721
00722     /* Set timer... */
00723     SELECT_TIMER("MODULE_TURBDIFF", NVTX_GPU);
00724
00725 #ifdef _OPENACC
00726 #pragma acc data present(ctl,atm,dt,rs)
00727 #pragma acc parallel loop independent gang vector

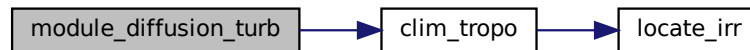
```

```

00728 #else
00729 #pragma omp parallel for default(shared)
00730 #endif
00731 for (int ip = 0; ip < atm->np; ip++)
00732     if (dt[ip] != 0) {
00733
00734         /* Get tropopause pressure... */
00735         double pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00736
00737         /* Get weighting factor... */
00738         double w;
00739         double p1 = pt * 0.866877899;
00740         double p0 = pt / 0.866877899;
00741         if (atm->p[ip] > p0)
00742             w = 1;
00743         else if (atm->p[ip] < p1)
00744             w = 0;
00745         else
00746             w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00747
00748         /* Set diffusivity... */
00749         double dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00750         double dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00751
00752         /* Horizontal turbulent diffusion... */
00753         if (dx > 0) {
00754             double sigma = sqrt(2.0 * dx * fabs(dt[ip]));
00755             atm->lon[ip] += DX2DEG(rs[3 * ip] * sigma / 1000., atm->lat[ip]);
00756             atm->lat[ip] += DY2DEG(rs[3 * ip + 1] * sigma / 1000.);
00757         }
00758
00759         /* Vertical turbulent diffusion... */
00760         if (dz > 0) {
00761             double sigma = sqrt(2.0 * dz * fabs(dt[ip]));
00762             atm->p[ip]
00763                 += DZ2DP(rs[3 * ip + 2] * sigma / 1000., atm->p[ip]);
00764         }
00765     }
00766 }

```

Here is the call graph for this function:



**5.37.2.6 module\_dry\_deposition()** void module\_dry\_deposition (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )

```

Calculate dry deposition.

Definition at line 770 of file trac.c.

```

00775     {
00776
00777         /* Set timer... */
00778         SELECT_TIMER("MODULE_DRYDEPO", NVTX_GPU);
00779
00780         /* Width of the surface layer [hPa]. */
00781         const double dp = 30.;
00782
00783         /* Check quantity flags... */

```

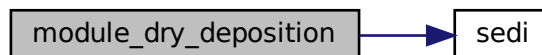


```

00784     if (ctl->qnt_m < 0)
00785         ERRMSG("Module needs quantity mass!");
00786
00787 #ifdef _OPENACC
00788 #pragma acc data present(ctl,met0,met1,atm,dt)
00789 #pragma acc parallel loop independent gang vector
00790 #else
00791 #pragma omp parallel for default(shared)
00792 #endif
00793     for (int ip = 0; ip < atm->np; ip++)
00794         if (dt[ip] != 0) {
00795
00796             double ps, t, v_dep;
00797
00798             /* Get surface pressure... */
00799             INTPOL_INIT;
00800             INTPOL_2D(ps, 1);
00801
00802             /* Check whether particle is above the surface layer... */
00803             if (atm->p[ip] < ps - dp)
00804                 continue;
00805
00806             /* Set width of surface layer... */
00807             double dz = 1000. * (Z(ps - dp) - Z(ps));
00808
00809             /* Calculate sedimentation velocity for particles... */
00810             if (ctl->qnt_r >= 0 && ctl->qnt_rho >= 0) {
00811
00812                 /* Get temperature... */
00813                 INTPOL_3D(t, 1);
00814
00815                 /* Set deposition velocity... */
00816                 v_dep = sedi(atm->p[ip], t, atm->q[ctl->qnt_r][ip],
00817                             atm->q[ctl->qnt_rho][ip]);
00818             }
00819
00820             /* Use explicit sedimentation velocity for gases... */
00821             else
00822                 v_dep = ctl->dry_depo[0];
00823
00824             /* Calculate loss of mass based on deposition velocity... */
00825             atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] * v_dep / dz);
00826         }
00827 }

```

Here is the call graph for this function:



**5.37.2.7 module\_isosurf\_init()** void module\_isosurf\_init (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    cache_t * cache )

```

Initialize isosurface module.

Definition at line 831 of file [trac.c](#).

```

00836     {
00837

```

```

00838 FILE *in;
00839
00840 char line[LEN];
00841
00842 double t;
00843
00844 /* Set timer... */
00845 SELECT_TIMER("MODULE_ISOSURF", NVTX_GPU);
00846
00847 /* Init... */
00848 INTPOL_INIT;
00849
00850 /* Save pressure... */
00851 if (ctl->isosurf == 1)
00852     for (int ip = 0; ip < atm->np; ip++)
00853         cache->iso_var[ip] = atm->p[ip];
00854
00855 /* Save density... */
00856 else if (ctl->isosurf == 2)
00857     for (int ip = 0; ip < atm->np; ip++) {
00858         INTPOL_3D(t, 1);
00859         cache->iso_var[ip] = atm->p[ip] / t;
00860     }
00861
00862 /* Save potential temperature... */
00863 else if (ctl->isosurf == 3)
00864     for (int ip = 0; ip < atm->np; ip++) {
00865         INTPOL_3D(t, 1);
00866         cache->iso_var[ip] = THETA(atm->p[ip], t);
00867     }
00868
00869 /* Read balloon pressure data... */
00870 else if (ctl->isosurf == 4) {
00871
00872     /* Write info... */
00873     printf("Read balloon pressure data: %s\n", ctl->balloon);
00874
00875     /* Open file... */
00876     if (!(in = fopen(ctl->balloon, "r")))
00877         ERRMSG("Cannot open file!");
00878
00879     /* Read pressure time series... */
00880     while (fgets(line, LEN, in))
00881         if (sscanf(line, "%lg %lg", &(cache->iso_ts[cache->iso_n]),
00882                 &(cache->iso_ps[cache->iso_n])) == 2)
00883             if ((++cache->iso_n) > NP)
00884                 ERRMSG("Too many data points!");
00885
00886     /* Check number of points... */
00887     if (cache->iso_n < 1)
00888         ERRMSG("Could not read any data!");
00889
00890     /* Close file... */
00891     fclose(in);
00892 }
00893 }

```

**5.37.2.8 module\_isosurf()** void module\_isosurf (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    cache_t * cache )

```

Force air parcels to stay on isosurface.

Definition at line 897 of file [trac.c](#).

```

00902     {
00903
00904     /* Set timer... */
00905     SELECT_TIMER("MODULE_ISOSURF", NVTX_GPU);
00906
00907     #ifdef _OPENACC
00908     #pragma acc data present(ctl,met0,met1,atm,cache)
00909     #pragma acc parallel loop independent gang vector
00910     #else
00911     #pragma omp parallel for default(shared)

```

```

00912 #endif
00913     for (int ip = 0; ip < atm->np; ip++) {
00914
00915         double t;
00916
00917         /* Init... */
00918         INTPOL_INIT;
00919
00920         /* Restore pressure... */
00921         if (ctl->isosurf == 1)
00922             atm->p[ip] = cache->iso_var[ip];
00923
00924         /* Restore density... */
00925         else if (ctl->isosurf == 2) {
00926             INTPOL_3D(t, 1);
00927             atm->p[ip] = cache->iso_var[ip] * t;
00928         }
00929
00930         /* Restore potential temperature... */
00931         else if (ctl->isosurf == 3) {
00932             INTPOL_3D(t, 1);
00933             atm->p[ip] = 1000. * pow(cache->iso_var[ip] / t, -1. / 0.286);
00934         }
00935
00936         /* Interpolate pressure... */
00937         else if (ctl->isosurf == 4) {
00938             if (atm->time[ip] <= cache->iso_ts[0])
00939                 atm->p[ip] = cache->iso_ps[0];
00940             else if (atm->time[ip] >= cache->iso_ts[cache->iso_n - 1])
00941                 atm->p[ip] = cache->iso_ps[cache->iso_n - 1];
00942             else {
00943                 int idx = locate_irr(cache->iso_ts, cache->iso_n, atm->time[ip]);
00944                 atm->p[ip] = LIN(cache->iso_ts[idx], cache->iso_ps[idx],
00945                               cache->iso_ts[idx + 1], cache->iso_ps[idx + 1],
00946                               atm->time[ip]);
00947             }
00948         }
00949     }
00950 }

```

Here is the call graph for this function:



```

5.37.2.9 module_meteo() void module_meteo (
    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm )

```

Interpolate meteorological data for air parcel positions.

Definition at line 954 of file [trac.c](#).

```

00958     {
00959
00960         /* Set timer... */
00961         SELECT_TIMER("MODULE_METEO", NVTX_GPU);
00962
00963         /* Check quantity flags... */
00964         if (ctl->qnt_tsts >= 0)
00965             if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
00966                 ERRMSG("Need T_ice and T_NAT to calculate T_STS!");

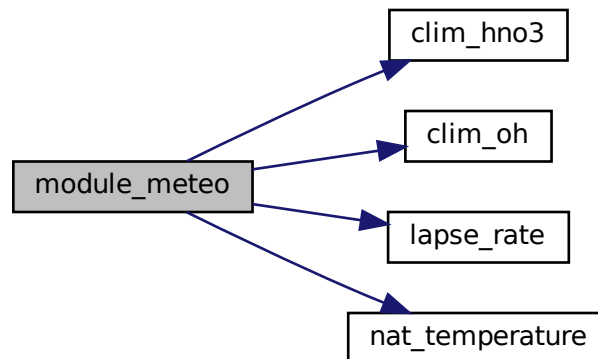
```

```

00967
00968 #ifndef _OPENACC
00969 #pragma acc data present(ctl,met0,met1,atm)
00970 #pragma acc parallel loop independent gang vector
00971 #else
00972 #pragma omp parallel for default(shared)
00973 #endif
00974     for (int ip = 0; ip < atm->np; ip++) {
00975
00976         double ps, ts, zs, us, vs, pt, pc, cl, plcl, plfc, pel, cape, pv, t, tt,
00977             u, v, w, h2o, h2ot, o3, lwc, iwc, z, zt;
00978
00979         /* Interpolate meteorological data... */
00980         INTPOL_INIT;
00981         INTPOL_TIME_ALL(atm->time[ip], atm->p[ip], atm->lon[ip], atm->lat[ip]);
00982
00983         /* Set quantities... */
00984         ATM_SET(qnt_ps, ps);
00985         ATM_SET(qnt_ts, ts);
00986         ATM_SET(qnt_zs, zs);
00987         ATM_SET(qnt_us, us);
00988         ATM_SET(qnt_vs, vs);
00989         ATM_SET(qnt_pt, pt);
00990         ATM_SET(qnt_tt, tt);
00991         ATM_SET(qnt_zt, zt);
00992         ATM_SET(qnt_h2ot, h2ot);
00993         ATM_SET(qnt_p, atm->p[ip]);
00994         ATM_SET(qnt_z, z);
00995         ATM_SET(qnt_t, t);
00996         ATM_SET(qnt_u, u);
00997         ATM_SET(qnt_v, v);
00998         ATM_SET(qnt_w, w);
00999         ATM_SET(qnt_h2o, h2o);
01000         ATM_SET(qnt_o3, o3);
01001         ATM_SET(qnt_lwc, lwc);
01002         ATM_SET(qnt_iwc, iwc);
01003         ATM_SET(qnt_pc, pc);
01004         ATM_SET(qnt_cl, cl);
01005         ATM_SET(qnt_plcl, plcl);
01006         ATM_SET(qnt_plfc, plfc);
01007         ATM_SET(qnt_pel, pel);
01008         ATM_SET(qnt_cape, cape);
01009         ATM_SET(qnt_hno3, clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip]));
01010         ATM_SET(qnt_oh, clim_oh(atm->time[ip], atm->lat[ip], atm->p[ip]));
01011         ATM_SET(qnt_vh, sqrt(u * u + v * v));
01012         ATM_SET(qnt_vz, -1e3 * H0 / atm->p[ip] * w);
01013         ATM_SET(qnt_psat, PSAT(t));
01014         ATM_SET(qnt_psice, PSICE(t));
01015         ATM_SET(qnt_pw, PW(atm->p[ip], h2o));
01016         ATM_SET(qnt_sh, SH(h2o));
01017         ATM_SET(qnt_rh, RH(atm->p[ip], t, h2o));
01018         ATM_SET(qnt_rhice, RHICE(atm->p[ip], t, h2o));
01019         ATM_SET(qnt_theta, THETA(atm->p[ip], t));
01020         ATM_SET(qnt_tvirt, TVIRT(t, h2o));
01021         ATM_SET(qnt_lapse, lapse_rate(t, h2o));
01022         ATM_SET(qnt_pv, pv);
01023         ATM_SET(qnt_tdew, TDEW(atm->p[ip], h2o));
01024         ATM_SET(qnt_tice, TICE(atm->p[ip], h2o));
01025         ATM_SET(qnt_tnat,
01026             nat_temperature(atm->p[ip], h2o,
01027                 clim_hno3(atm->time[ip], atm->lat[ip],
01028                     atm->p[ip]) * 1e-9));
01029         ATM_SET(qnt_tsts,
01030             0.5 * (atm->q[ctl->qnt_tice][ip] + atm->q[ctl->qnt_tnat][ip]));
01031     }
01032 }

```

Here is the call graph for this function:



**5.37.2.10 module\_position()** void module\_position (

```

    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )

```

Check position of air parcels.

Definition at line 1036 of file trac.c.

```

01040     {
01041
01042     /* Set timer... */
01043     SELECT_TIMER("MODULE_POSITION", NVTX_GPU);
01044
01045     #ifdef _OPENACC
01046     #pragma acc data present(met0,met1,atm,dt)
01047     #pragma acc parallel loop independent gang vector
01048     #else
01049     #pragma omp parallel for default(shared)
01050     #endif
01051     for (int ip = 0; ip < atm->np; ip++)
01052     if (dt[ip] != 0) {
01053
01054         /* Calculate modulo... */
01055         atm->lon[ip] = FMOD(atm->lon[ip], 360.);
01056         atm->lat[ip] = FMOD(atm->lat[ip], 360.);
01057
01058         /* Check latitude... */
01059         while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
01060             if (atm->lat[ip] > 90) {
01061                 atm->lat[ip] = 180 - atm->lat[ip];
01062                 atm->lon[ip] += 180;
01063             }
01064             if (atm->lat[ip] < -90) {
01065                 atm->lat[ip] = -180 - atm->lat[ip];
01066                 atm->lon[ip] += 180;
01067             }
01068         }
01069
01070         /* Check longitude... */
01071         while (atm->lon[ip] < -180)
01072             atm->lon[ip] += 360;
01073         while (atm->lon[ip] >= 180)
01074             atm->lon[ip] -= 360;

```

```

01075
01076     /* Check pressure... */
01077     if (atm->p[ip] < met0->p[met0->np - 1])
01078         atm->p[ip] = met0->p[met0->np - 1];
01079     else if (atm->p[ip] > 300.) {
01080         double ps;
01081         INTPOL_INIT;
01082         INTPOL_2D(ps, 1);
01083         if (atm->p[ip] > ps)
01084             atm->p[ip] = ps;
01085     }
01086 }
01087 }

```

**5.37.2.11 module\_rng\_init()** void module\_rng\_init (  
void )

Initialize random number generator...

Definition at line 1091 of file [trac.c](#).

```

01092     {
01093
01094         /* Set timer... */
01095         SELECT_TIMER("MODULE_RNG", NVTX_GPU);
01096
01097         /* Initialize random number generator... */
01098 #ifdef _OPENACC
01099
01100         if (curandCreateGenerator(&rng, CURAND_RNG_PSEUDO_DEFAULT)
01101             != CURAND_STATUS_SUCCESS)
01102             ERRMSG("Cannot create random number generator!");
01103         if (curandSetStream(rng, (cudaStream_t) acc_get_cuda_stream(acc_async_sync))
01104             != CURAND_STATUS_SUCCESS)
01105             ERRMSG("Cannot set stream for random number generator!");
01106 #else
01107
01108         gsl_rng_env_setup();
01109         if (omp_get_max_threads() > NTHREADS)
01110             ERRMSG("Too many threads!");
01111         for (int i = 0; i < NTHREADS; i++) {
01112             rng[i] = gsl_rng_alloc(gsl_rng_default);
01113             gsl_rng_set(rng[i], gsl_rng_default_seed + (long unsigned) i);
01114         }
01115 #endif
01116 }
01117 }
01118 }

```

**5.37.2.12 module\_rng()** void module\_rng (  
double \* rs,  
size\_t n,  
int method )

Generate random numbers.

Definition at line 1122 of file [trac.c](#).

```

01125     {
01126
01127         /* Set timer... */
01128         SELECT_TIMER("MODULE_RNG", NVTX_GPU);
01129
01130 #ifdef _OPENACC
01131
01132 #pragma acc host_data use_device(rs)
01133     {
01134         /* Uniform distribution... */
01135         if (method == 0) {
01136             if (curandGenerateUniform(rng, rs, n)
01137                 != CURAND_STATUS_SUCCESS)
01138                 ERRMSG("Cannot create random numbers!");

```

```

01139     }
01140
01141     /* Normal distribution... */
01142     else if (method == 1) {
01143         if (curandGenerateNormalDouble(rng, rs, n, 0.0, 1.0)
01144             != CURAND_STATUS_SUCCESS)
01145             ERRMSG("Cannot create random numbers!");
01146     }
01147 }
01148
01149 #else
01150
01151     /* Uniform distribution... */
01152     if (method == 0) {
01153 #pragma omp parallel for default(shared)
01154         for (size_t i = 0; i < n; ++i)
01155             rs[i] = gsl_rng_uniform(rng[omp_get_thread_num()]);
01156     }
01157
01158     /* Normal distribution... */
01159     else if (method == 1) {
01160 #pragma omp parallel for default(shared)
01161         for (size_t i = 0; i < n; ++i)
01162             rs[i] = gsl_ran_gaussian_ziggurat(rng[omp_get_thread_num()], 1.0);
01163     }
01164 #endif
01165
01166 }

```

**5.37.2.13 module\_sedi()** void module\_sedi (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )

```

Calculate sedimentation of air parcels.

Definition at line 1170 of file [trac.c](#).

```

01175     {
01176
01177     /* Set timer... */
01178     SELECT_TIMER("MODULE_SEDI", NVTX_GPU);
01179
01180 #ifdef _OPENACC
01181 #pragma acc data present(ctl,met0,met1,atm,dt)
01182 #pragma acc parallel loop independent gang vector
01183 #else
01184 #pragma omp parallel for default(shared)
01185 #endif
01186     for (int ip = 0; ip < atm->np; ip++)
01187         if (dt[ip] != 0) {
01188
01189             /* Get temperature... */
01190             double t;
01191             INTPOL_INIT;
01192             INTPOL_3D(t, 1);
01193
01194             /* Sedimentation velocity... */
01195             double v_s = sedi(atm->p[ip], t, atm->q[ctl->qnt_r][ip],
01196                             atm->q[ctl->qnt_rho][ip]);
01197
01198             /* Calculate pressure change... */
01199             atm->p[ip] += DZ2DP(v_s * dt[ip] / 1000., atm->p[ip]);
01200         }
01201 }

```

Here is the call graph for this function:



#### 5.37.2.14 module\_oh\_chem() void module\_oh\_chem (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )

```

Calculate OH chemistry.

Definition at line 1205 of file [trac.c](#).

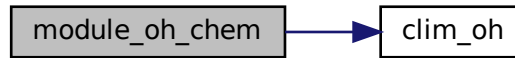
```

01210     {
01211
01212     /* Set timer... */
01213     SELECT_TIMER("MODULE_OHCHEM", NVTX_GPU);
01214
01215     /* Check quantity flags... */
01216     if (ctl->qnt_m < 0)
01217         ERRMSG("Module needs quantity mass!");
01218
01219     #ifdef _OPENACC
01220     #pragma acc data present(ctl,met0,met1,atm,dt)
01221     #pragma acc parallel loop independent gang vector
01222     #else
01223     #pragma omp parallel for default(shared)
01224     #endif
01225     for (int ip = 0; ip < atm->np; ip++)
01226     if (dt[ip] != 0) {
01227
01228         /* Get temperature... */
01229         double t;
01230         INTPOL_INIT;
01231         INTPOL_3D(t, 1);
01232
01233         /* Calculate molecular density... */
01234         double M = 7.243e21 * (atm->p[ip] / P0) / t;
01235
01236         /* Calculate rate coefficient for X + OH + M -> XOH + M
01237          (JPL Publication 15-10) ... */
01238         double k0 = ctl->oh_chem[0] *
01239             (ctl->oh_chem[1] > 0 ? pow(t / 300., -ctl->oh_chem[1]) : 1.);
01240         double ki = ctl->oh_chem[2] *
01241             (ctl->oh_chem[3] > 0 ? pow(t / 300., -ctl->oh_chem[3]) : 1.);
01242         double c = log10(k0 * M / ki);
01243         double k = k0 * M / (1. + k0 * M / ki) * pow(0.6, 1. / (1. + c * c));
01244
01245         /* Calculate exponential decay... */
01246         atm->q[ctl->qnt_m][ip] *=
01247             exp(-dt[ip] * k * clim_oh(atm->time[ip], atm->lat[ip], atm->p[ip]));
01248     }
01249 }

```



Here is the call graph for this function:



### 5.37.2.15 module\_wet\_deposition() void module\_wet\_deposition (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )

```

Calculate wet deposition.

Definition at line 1253 of file [trac.c](#).

```

01258     {
01259
01260     /* Set timer... */
01261     SELECT_TIMER("MODULE_WETDEPO", NVTX_GPU);
01262
01263     /* Check quantity flags... */
01264     if (ctl->qnt_m < 0)
01265         ERRMSG("Module needs quantity mass!");
01266
01267 #ifdef _OPENACC
01268 #pragma acc data present(ctl,met0,met1,atm,dt)
01269 #pragma acc parallel loop independent gang vector
01270 #else
01271 #pragma omp parallel for default(shared)
01272 #endif
01273     for (int ip = 0; ip < atm->np; ip++)
01274         if (dt[ip] != 0) {
01275
01276             double cl, dz, h, lambda = 0, t, iwc, lwc, pc;
01277
01278             int inside;
01279
01280             /* Check whether particle is below cloud top... */
01281             INTPOL_INIT;
01282             INTPOL_2D(pc, 1);
01283             if (!isfinite(pc) || atm->p[ip] <= pc)
01284                 continue;
01285
01286             /* Estimate precipitation rate (Pisso et al., 2019)... */
01287             INTPOL_2D(cl, 0);
01288             double Is = pow(2. * cl, 1. / 0.36);
01289             if (Is < 0.01)
01290                 continue;
01291
01292             /* Check whether particle is inside or below cloud... */
01293             INTPOL_3D(lwc, 1);
01294             INTPOL_3D(iwc, 0);
01295             inside = (iwc > 0 || lwc > 0);
01296
01297             /* Calculate in-cloud scavenging coefficient... */
01298             if (inside) {
01299
01300                 /* Use exponential dependency for particles... */
01301                 if (ctl->wet_depo[0] > 0)
01302                     lambda = ctl->wet_depo[0] * pow(Is, ctl->wet_depo[1]);
01303
01304                 /* Use Henry's law for gases... */
01305                 else if (ctl->wet_depo[2] > 0) {

```

```

01306
01307     /* Get temperature... */
01308     INTPOL_3D(t, 0);
01309
01310     /* Get Henry's constant (Sander, 2015)... */
01311     h = ctl->wet_depo[2]
01312       * exp(ctl->wet_depo[3] * (1. / t - 1. / 298.15));
01313
01314     /* Estimate depth of cloud layer... */
01315     dz = 1e3 * Z(pc);
01316
01317     /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
01318     lambda = h * RI * t * Is / 3.6e6 * dz;
01319   }
01320 }
01321
01322 /* Calculate below-cloud scavenging coefficient... */
01323 else {
01324
01325     /* Use exponential dependency for particles... */
01326     if (ctl->wet_depo[4] > 0)
01327         lambda = ctl->wet_depo[4] * pow(Is, ctl->wet_depo[5]);
01328
01329     /* Use Henry's law for gases... */
01330     else if (ctl->wet_depo[6] > 0) {
01331
01332         /* Get temperature... */
01333         INTPOL_3D(t, 0);
01334
01335         /* Get Henry's constant (Sander, 2015)... */
01336         h = ctl->wet_depo[6]
01337           * exp(ctl->wet_depo[7] * (1. / t - 1. / 298.15));
01338
01339         /* Estimate depth of cloud layer... */
01340         dz = 1e3 * Z(pc);
01341
01342         /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
01343         lambda = h * RI * t * Is / 3.6e6 * dz;
01344     }
01345 }
01346
01347 /* Calculate exponential decay of mass... */
01348 atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] * lambda);
01349 }
01350 }

```

#### 5.37.2.16 write\_output() 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 1354 of file [trac.c](#).

```

01360     {
01361
01362     char filename[2 * LEN];
01363
01364     double r;
01365
01366     int year, mon, day, hour, min, sec, updated = 0;
01367
01368     /* Get time... */
01369     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01370
01371     /* Write atmospheric data... */
01372     if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01373         if (!updated) {
01374             #ifndef _OPENACC
01375                 SELECT_TIMER("UPDATE_HOST", NVTX_D2H);
01376             #pragma acc update host (atm[:1])
01377             #endif
01378             updated = 1;

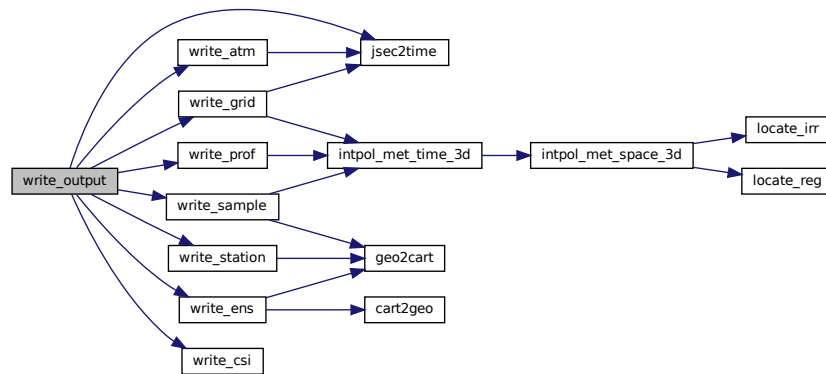
```

```

01379     }
01380     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.tab",
01381             dirname, ctl->atm_basename, year, mon, day, hour, min);
01382     write_atm(filename, ctl, atm, t);
01383 }
01384
01385 /* Write gridded data... */
01386 if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01387     if (!updated) {
01388 #ifdef _OPENACC
01389         SELECT_TIMER("UPDATE_HOST", NVTX_D2H);
01390 #pragma acc update host(atm[:1])
01391 #endif
01392         updated = 1;
01393     }
01394     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.tab",
01395             dirname, ctl->grid_basename, year, mon, day, hour, min);
01396     write_grid(filename, ctl, met0, met1, atm, t);
01397 }
01398
01399 /* Write CSI data... */
01400 if (ctl->csi_basename[0] != '-') {
01401     if (!updated) {
01402 #ifdef _OPENACC
01403         SELECT_TIMER("UPDATE_HOST", NVTX_D2H);
01404 #pragma acc update host(atm[:1])
01405 #endif
01406         updated = 1;
01407     }
01408     sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01409     write_csi(filename, ctl, atm, t);
01410 }
01411
01412 /* Write ensemble data... */
01413 if (ctl->ens_basename[0] != '-') {
01414     if (!updated) {
01415 #ifdef _OPENACC
01416         SELECT_TIMER("UPDATE_HOST", NVTX_D2H);
01417 #pragma acc update host(atm[:1])
01418 #endif
01419         updated = 1;
01420     }
01421     sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01422     write_ens(filename, ctl, atm, t);
01423 }
01424
01425 /* Write profile data... */
01426 if (ctl->prof_basename[0] != '-') {
01427     if (!updated) {
01428 #ifdef _OPENACC
01429         SELECT_TIMER("UPDATE_HOST", NVTX_D2H);
01430 #pragma acc update host(atm[:1])
01431 #endif
01432         updated = 1;
01433     }
01434     sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01435     write_prof(filename, ctl, met0, met1, atm, t);
01436 }
01437
01438 /* Write sample data... */
01439 if (ctl->sample_basename[0] != '-') {
01440     if (!updated) {
01441 #ifdef _OPENACC
01442         SELECT_TIMER("UPDATE_HOST", NVTX_D2H);
01443 #pragma acc update host(atm[:1])
01444 #endif
01445         updated = 1;
01446     }
01447     sprintf(filename, "%s/%s.tab", dirname, ctl->sample_basename);
01448     write_sample(filename, ctl, met0, met1, atm, t);
01449 }
01450
01451 /* Write station data... */
01452 if (ctl->stat_basename[0] != '-') {
01453     if (!updated) {
01454 #ifdef _OPENACC
01455         SELECT_TIMER("UPDATE_HOST", NVTX_D2H);
01456 #pragma acc update host(atm[:1])
01457 #endif
01458         updated = 1;
01459     }
01460     sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01461     write_station(filename, ctl, atm, t);
01462 }
01463 }

```

Here is the call graph for this function:



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

Definition at line 165 of file [trac.c](#).

```

00167     {
00168
00169     ctl_t ctl;
00170
00171     atm_t *atm;
00172
00173     cache_t *cache;
00174
00175     met_t *met0, *met1;
00176
00177     FILE *dirlist;
00178
00179     char dirname[LEN], filename[2 * LEN];
00180
00181     double *dt, *rs, t;
00182
00183     int num_devices = 0, ntask = -1, rank = 0, size = 1;
00184
00185     /* Start timers... */
00186     START_TIMERS;
00187
00188     /* Initialize MPI... */
00189     #ifdef MPI
00190     SELECT_TIMER("MPI_INIT", NVTX_CPU);
00191     MPI_Init(&argc, &argv);
00192     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00193     MPI_Comm_size(MPI_COMM_WORLD, &size);
00194     #endif
00195
00196     /* Initialize GPUs... */
00197     #ifdef _OPENACC
00198     SELECT_TIMER("ACC_INIT", NVTX_GPU);
00199     acc_device_t device_type = acc_get_device_type();
00200     num_devices = acc_get_num_devices(acc_device_nvidia);
00201     int device_num = rank % num_devices;
00202     acc_set_device_num(device_num, acc_device_nvidia);
00203     acc_init(device_type);
00204     #endif
00205
00206     /* Check arguments... */
00207     if (argc < 4)
00208         ERRMSG("Give parameters: <dirlist> <ctl> <atm_in>");
00209
00210     /* Open directory list... */
00211     if (!(dirlist = fopen(argv[1], "r")))
00212         ERRMSG("Cannot open directory list!");
  
```

```

00213
00214 /* Loop over directories... */
00215 while (fscanf(dirlist, "%s", dirname) != EOF) {
00216
00217     /* MPI parallelization... */
00218     if ((++ntask) % size != rank)
00219         continue;
00220
00221     /* -----
00222      Initialize model run...
00223     ----- */
00224
00225     /* Allocate... */
00226     SELECT_TIMER("ALLOC", NVTX_CPU);
00227     ALLOC(atm, atm_t, 1);
00228     ALLOC(cache, cache_t, 1);
00229     ALLOC(met0, met_t, 1);
00230     ALLOC(met1, met_t, 1);
00231     ALLOC(dt, double,
00232           NP);
00233     ALLOC(rs, double,
00234           3 * NP);
00235
00236     /* Create data region on GPUs... */
00237 #ifdef _OPENACC
00238     SELECT_TIMER("CREATE_DATA_REGION", NVTX_GPU);
00239 #pragma acc enter data create(atm[:1],cache[:1],ctl,met0[:1],met1[:1],dt[:NP],rs[:3*NP])
00240 #endif
00241
00242     /* Read control parameters... */
00243     sprintf(filename, "%s/%s", dirname, argv[2]);
00244     read_ctl(filename, argc, argv, &ctl);
00245
00246     /* Read atmospheric data... */
00247     sprintf(filename, "%s/%s", dirname, argv[3]);
00248     if (!read_atm(filename, &ctl, atm))
00249         ERRMSG("Cannot open file!");
00250
00251     /* Set start time... */
00252     SELECT_TIMER("TIMESTEPS", NVTX_CPU);
00253     if (ctl.direction == 1) {
00254         ctl.t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00255         if (ctl.t_stop > 1e99)
00256             ctl.t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00257     } else {
00258         ctl.t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00259         if (ctl.t_stop > 1e99)
00260             ctl.t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00261     }
00262
00263     /* Check time interval... */
00264     if (ctl.direction * (ctl.t_stop - ctl.t_start) <= 0)
00265         ERRMSG("Nothing to do! Check T_STOP and DIRECTION!");
00266
00267     /* Round start time... */
00268     if (ctl.direction == 1)
00269         ctl.t_start = floor(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00270     else
00271         ctl.t_start = ceil(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00272
00273     /* Update GPU... */
00274 #ifdef _OPENACC
00275     SELECT_TIMER("UPDATE_DEVICE", NVTX_H2D);
00276 #pragma acc update device(atm[:1],ctl)
00277 #endif
00278
00279     /* Initialize random number generator... */
00280     module_rng_init();
00281
00282     /* Initialize meteorological data... */
00283     get_met(&ctl, ctl.t_start, &met0, &met1);
00284     if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00285         WARN("Violation of CFL criterion! Check DT_MOD!");
00286
00287     /* Initialize isosurface... */
00288     if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00289         module_isosurf_init(&ctl, met0, met1, atm, cache);
00290
00291     /* Update GPU... */
00292 #ifdef _OPENACC
00293     SELECT_TIMER("UPDATE_DEVICE", NVTX_H2D);
00294 #pragma acc update device(cache[:1])
00295 #endif
00296
00297     /* -----
00298      Loop over timesteps...
00299     ----- */

```

```

00300
00301      /* Loop over timesteps... */
00302      for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
00303           t += ctl.direction * ctl.dt_mod) {
00304
00305          /* Adjust length of final time step... */
00306          if (ctl.direction * (t - ctl.t_stop) > 0)
00307              t = ctl.t_stop;
00308
00309          /* Set time steps for air parcels... */
00310          SELECT_TIMER("TIMESTEPS", NVTX_GPU);
00311 #ifdef _OPENACC
00312 #pragma acc parallel loop independent gang vector present(ctl,atm,atm->time,dt)
00313 #endif
00314          for (int ip = 0; ip < atm->np; ip++) {
00315              double atmtime = atm->time[ip];
00316              double tstart = ctl.t_start;
00317              double tstop = ctl.t_stop;
00318              int dir = ctl.direction;
00319              if ((dir * (atmtime - tstart) >= 0 && dir * (atmtime - tstop) <= 0
00320                  && dir * (atmtime - t) < 0))
00321                  dt[ip] = t - atmtime;
00322              else
00323                  dt[ip] = 0;
00324          }
00325
00326          /* Get meteorological data... */
00327          if (t != ctl.t_start)
00328              get_met(&ctl, t, &met0, &met1);
00329
00330          /* Check initial positions... */
00331          module_position(met0, met1, atm, dt);
00332
00333          /* Advection... */
00334          module_advection(met0, met1, atm, dt);
00335
00336          /* Turbulent diffusion... */
00337          if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
00338              || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0) {
00339              module_rng(rs, 3 * (size_t) atm->np, 1);
00340              module_diffusion_turb(&ctl, atm, dt, rs);
00341          }
00342
00343          /* Mesoscale diffusion... */
00344          if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0) {
00345              module_rng(rs, 3 * (size_t) atm->np, 1);
00346              module_diffusion_meso(&ctl, met0, met1, atm, cache, dt, rs);
00347          }
00348
00349          /* Convection... */
00350          if (ctl.conv_cape >= 0) {
00351              module_rng(rs, (size_t) atm->np, 0);
00352              module_convection(&ctl, met0, met1, atm, dt, rs);
00353          }
00354
00355          /* Sedimentation... */
00356          if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0)
00357              module_sedi(&ctl, met0, met1, atm, dt);
00358
00359          /* Isosurface... */
00360          if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00361              module_isosurf(&ctl, met0, met1, atm, cache);
00362
00363          /* Check final positions... */
00364          module_position(met0, met1, atm, dt);
00365
00366          /* Interpolate meteorological data... */
00367          if (ctl.met_dt_out > 0
00368              && (ctl.met_dt_out < ctl.dt_mod || fmod(t, ctl.met_dt_out) == 0))
00369              module_meteo(&ctl, met0, met1, atm);
00370
00371          /* Decay of particle mass... */
00372          if (ctl.tdec_trop > 0 && ctl.tdec_strat > 0)
00373              module_decay(&ctl, atm, dt);
00374
00375          /* OH chemistry... */
00376          if (ctl.oh_chem[0] > 0 && ctl.oh_chem[2] > 0)
00377              module_oh_chem(&ctl, met0, met1, atm, dt);
00378
00379          /* Dry deposition... */
00380          if (ctl.dry_depo[0] > 0)
00381              module_dry_deposition(&ctl, met0, met1, atm, dt);
00382
00383          /* Wet deposition... */
00384          if ((ctl.wet_depo[0] > 0 || ctl.wet_depo[2] > 0)
00385              && (ctl.wet_depo[4] > 0 || ctl.wet_depo[6] > 0))
00386              module_wet_deposition(&ctl, met0, met1, atm, dt);

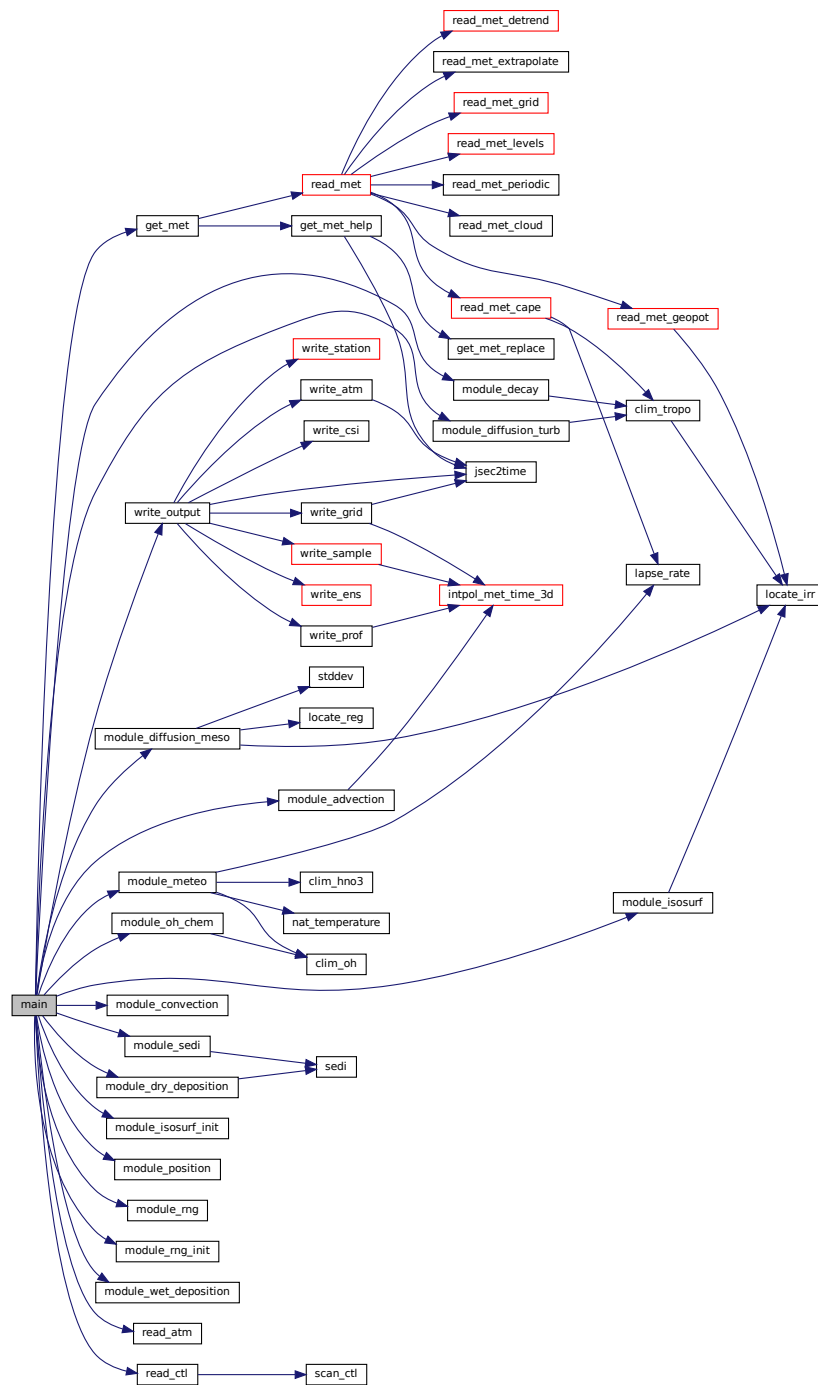
```

```

00387
00388     /* Write output... */
00389     write_output(dirname, &ctl, met0, met1, atm, t);
00390 }
00391
00392 /* -----
00393     Finalize model run...
00394     ----- */
00395
00396 /* Report problem size... */
00397 printf("SIZE_NP = %d\n", atm->np);
00398 printf("SIZE_MPI_TASKS = %d\n", size);
00399 printf("SIZE_OMP_THREADS = %d\n", omp_get_max_threads());
00400 printf("SIZE_ACC_DEVICES = %d\n", num_devices);
00401
00402 /* Report memory usage... */
00403 printf("MEMORY_ATM = %g MByte\n", sizeof(atm_t) / 1024. / 1024.);
00404 printf("MEMORY_CACHE = %g MByte\n", sizeof(cache_t) / 1024. / 1024.);
00405 printf("MEMORY_METEO = %g MByte\n", 2 * sizeof(met_t) / 1024. / 1024.);
00406 printf("MEMORY_DYNAMIC = %g MByte\n", (sizeof(met_t)
00407                                     + 4 * NP * sizeof(double)
00408                                     + EX * EY * EP * sizeof(float)) /
00409                                     1024. / 1024.);
00410 printf("MEMORY_STATIC = %g MByte\n", (EX * EY * sizeof(double)
00411                                     + EX * EY * EP * sizeof(float)
00412                                     + 4 * GX * GY * GZ * sizeof(double)
00413                                     + 2 * GX * GY * GZ * sizeof(int)
00414                                     + 2 * GX * GY * sizeof(double)
00415                                     + GX * GY * sizeof(int)) / 1024. /
00416                                     1024.);
00417
00418 /* Delete data region on GPUs... */
00419 #ifndef _OPENACC
00420     SELECT_TIMER("DELETE_DATA_REGION", NVTX_GPU);
00421 #pragma acc exit data delete(ctl,atm,cache,met0,met1,dt,rs)
00422 #endif
00423
00424 /* Free... */
00425 SELECT_TIMER("FREE", NVTX_CPU);
00426 free(atm);
00427 free(cache);
00428 free(met0);
00429 free(met1);
00430 free(dt);
00431 free(rs);
00432
00433 /* Report timers... */
00434 PRINT_TIMERS;
00435 }
00436
00437 /* Finalize MPI... */
00438 #ifdef MPI
00439     MPI_Finalize();
00440 #endif
00441
00442 /* Stop timers... */
00443 STOP_TIMERS;
00444
00445 return EXIT_SUCCESS;
00446 }

```

Here is the call graph for this function:



### 5.37.3 Variable Documentation

#### 5.37.3.1 rng static gsl\_rng \* rng

Definition at line 32 of file [trac.c](#).



## 5.38 trac.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
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028  Global variables...
00029  ----- */
00030
00031 #ifdef _OPENACC
00032 curandGenerator_t rng;
00033 #else
00034 static gsl_rng *rng[NTHREADS];
00035 #endif
00036
00037 /* -----
00038  Functions...
00039  ----- */
00040
00042 void module_advection(
00043     met_t * met0,
00044     met_t * met1,
00045     atm_t * atm,
00046     double *dt);
00047
00049 void module_convection(
00050     ctl_t * ctl,
00051     met_t * met0,
00052     met_t * met1,
00053     atm_t * atm,
00054     double *dt,
00055     double *rs);
00056
00058 void module_decay(
00059     ctl_t * ctl,
00060     atm_t * atm,
00061     double *dt);
00062
00064 void module_diffusion_meso(
00065     ctl_t * ctl,
00066     met_t * met0,
00067     met_t * met1,
00068     atm_t * atm,
00069     cache_t * cache,
00070     double *dt,
00071     double *rs);
00072
00074 void module_diffusion_turb(
00075     ctl_t * ctl,
00076     atm_t * atm,
00077     double *dt,
00078     double *rs);
00079
00081 void module_dry_deposition(
00082     ctl_t * ctl,
00083     met_t * met0,
00084     met_t * met1,
00085     atm_t * atm,
00086     double *dt);
00087
00089 void module_isosurf_init(
00090     ctl_t * ctl,
00091     met_t * met0,
00092     met_t * met1,
00093     atm_t * atm,
00094     cache_t * cache);
00095
00097 void module_isosurf(
00098     ctl_t * ctl,

```

```

00099     met_t * met0,
00100     met_t * met1,
00101     atm_t * atm,
00102     cache_t * cache);
00103
00105 void module_meteo(
00106     ctl_t * ctl,
00107     met_t * met0,
00108     met_t * met1,
00109     atm_t * atm);
00110
00112 void module_position(
00113     met_t * met0,
00114     met_t * met1,
00115     atm_t * atm,
00116     double *dt);
00117
00119 void module_rng_init(
00120     void);
00121
00123 void module_rng(
00124     double *rs,
00125     size_t n,
00126     int method);
00127
00129 void module_sedi(
00130     ctl_t * ctl,
00131     met_t * met0,
00132     met_t * met1,
00133     atm_t * atm,
00134     double *dt);
00135
00137 void module_oh_chem(
00138     ctl_t * ctl,
00139     met_t * met0,
00140     met_t * met1,
00141     atm_t * atm,
00142     double *dt);
00143
00145 void module_wet_deposition(
00146     ctl_t * ctl,
00147     met_t * met0,
00148     met_t * met1,
00149     atm_t * atm,
00150     double *dt);
00151
00153 void write_output(
00154     const char *dirname,
00155     ctl_t * ctl,
00156     met_t * met0,
00157     met_t * met1,
00158     atm_t * atm,
00159     double t);
00160
00161 /* -----
00162     Main...
00163     ----- */
00164
00165 int main(
00166     int argc,
00167     char *argv[]) {
00168
00169     ctl_t ctl;
00170
00171     atm_t *atm;
00172
00173     cache_t *cache;
00174
00175     met_t *met0, *met1;
00176
00177     FILE *dirlist;
00178
00179     char dirname[LEN], filename[2 * LEN];
00180
00181     double *dt, *rs, t;
00182
00183     int num_devices = 0, ntask = -1, rank = 0, size = 1;
00184
00185     /* Start timers... */
00186     START_TIMERS;
00187
00188     /* Initialize MPI... */
00189     #ifdef MPI
00190     SELECT_TIMER("MPI_INIT", NVTX_CPU);
00191     MPI_Init(&argc, &argv);
00192     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00193     MPI_Comm_size(MPI_COMM_WORLD, &size);

```

```

00194 #endif
00195
00196 /* Initialize GPUs... */
00197 #ifdef _OPENACC
00198     SELECT_TIMER("ACC_INIT", NVTX_GPU);
00199     acc_device_t device_type = acc_get_device_type();
00200     num_devices = acc_get_num_devices(acc_device_nvidia);
00201     int device_num = rank % num_devices;
00202     acc_set_device_num(device_num, acc_device_nvidia);
00203     acc_init(device_type);
00204 #endif
00205
00206 /* Check arguments... */
00207 if (argc < 4)
00208     ERRMSG("Give parameters: <dirlist> <ctl> <atm_in>");
00209
00210 /* Open directory list... */
00211 if (!(dirlist = fopen(argv[1], "r")))
00212     ERRMSG("Cannot open directory list!");
00213
00214 /* Loop over directories... */
00215 while (fscanf(dirlist, "%s", dirname) != EOF) {
00216
00217     /* MPI parallelization... */
00218     if ((++ntask) % size != rank)
00219         continue;
00220
00221     /* -----
00222        Initialize model run...
00223        ----- */
00224
00225     /* Allocate... */
00226     SELECT_TIMER("ALLOC", NVTX_CPU);
00227     ALLOC(atm, atm_t, 1);
00228     ALLOC(cache, cache_t, 1);
00229     ALLOC(met0, met_t, 1);
00230     ALLOC(met1, met_t, 1);
00231     ALLOC(dt, double,
00232           NP);
00233     ALLOC(rs, double,
00234           3 * NP);
00235
00236     /* Create data region on GPUs... */
00237 #ifdef _OPENACC
00238     SELECT_TIMER("CREATE_DATA_REGION", NVTX_GPU);
00239 #pragma acc enter data create(atm[:1],cache[:1],ctl,met0[:1],met1[:1],dt[:NP],rs[:3*NP])
00240 #endif
00241
00242     /* Read control parameters... */
00243     sprintf(filename, "%s/%s", dirname, argv[2]);
00244     read_ctl(filename, argc, argv, &ctl);
00245
00246     /* Read atmospheric data... */
00247     sprintf(filename, "%s/%s", dirname, argv[3]);
00248     if (!read_atm(filename, &ctl, atm))
00249         ERRMSG("Cannot open file!");
00250
00251     /* Set start time... */
00252     SELECT_TIMER("Timesteps", NVTX_CPU);
00253     if (ctl.direction == 1) {
00254         ctl.t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00255         if (ctl.t_stop > 1e99)
00256             ctl.t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00257     } else {
00258         ctl.t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00259         if (ctl.t_stop > 1e99)
00260             ctl.t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00261     }
00262
00263     /* Check time interval... */
00264     if (ctl.direction * (ctl.t_stop - ctl.t_start) <= 0)
00265         ERRMSG("Nothing to do! Check T_STOP and DIRECTION!");
00266
00267     /* Round start time... */
00268     if (ctl.direction == 1)
00269         ctl.t_start = floor(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00270     else
00271         ctl.t_start = ceil(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00272
00273     /* Update GPU... */
00274 #ifdef _OPENACC
00275     SELECT_TIMER("UPDATE_DEVICE", NVTX_H2D);
00276 #pragma acc update device(atm[:1],ctl)
00277 #endif
00278
00279     /* Initialize random number generator... */
00280     module_rng_init();

```

```

00281
00282 /* Initialize meteorological data... */
00283 get_met(&ctl, ctl.t_start, &met0, &met1);
00284 if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00285     WARN("Violation of CFL criterion! Check DT_MOD!");
00286
00287 /* Initialize isosurface... */
00288 if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00289     module_isosurf_init(&ctl, met0, met1, atm, cache);
00290
00291 /* Update GPU... */
00292 #ifndef _OPENACC
00293     SELECT_TIMER("UPDATE_DEVICE", NVTX_H2D);
00294 #pragma acc update device(cache[:1])
00295 #endif
00296
00297 /* -----
00298     Loop over timesteps...
00299     ----- */
00300
00301 /* Loop over timesteps... */
00302 for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
00303      t += ctl.direction * ctl.dt_mod) {
00304
00305     /* Adjust length of final time step... */
00306     if (ctl.direction * (t - ctl.t_stop) > 0)
00307         t = ctl.t_stop;
00308
00309     /* Set time steps for air parcels... */
00310     SELECT_TIMER("TIMESTEPS", NVTX_GPU);
00311 #ifndef _OPENACC
00312 #pragma acc parallel loop independent gang vector present(ctl,atm,atm->time,dt)
00313 #endif
00314     for (int ip = 0; ip < atm->np; ip++) {
00315         double atmtime = atm->time[ip];
00316         double tstart = ctl.t_start;
00317         double tstop = ctl.t_stop;
00318         int dir = ctl.direction;
00319         if ((dir * (atmtime - tstart) >= 0 && dir * (atmtime - tstop) <= 0
00320             && dir * (atmtime - t) < 0))
00321             dt[ip] = t - atmtime;
00322         else
00323             dt[ip] = 0;
00324     }
00325
00326     /* Get meteorological data... */
00327     if (t != ctl.t_start)
00328         get_met(&ctl, t, &met0, &met1);
00329
00330     /* Check initial positions... */
00331     module_position(met0, met1, atm, dt);
00332
00333     /* Advection... */
00334     module_advection(met0, met1, atm, dt);
00335
00336     /* Turbulent diffusion... */
00337     if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
00338         || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0) {
00339         module_rng(rs, 3 * (size_t) atm->np, 1);
00340         module_diffusion_turb(&ctl, atm, dt, rs);
00341     }
00342
00343     /* Mesoscale diffusion... */
00344     if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0) {
00345         module_rng(rs, 3 * (size_t) atm->np, 1);
00346         module_diffusion_meso(&ctl, met0, met1, atm, cache, dt, rs);
00347     }
00348
00349     /* Convection... */
00350     if (ctl.conv_cape >= 0) {
00351         module_rng(rs, (size_t) atm->np, 0);
00352         module_convection(&ctl, met0, met1, atm, dt, rs);
00353     }
00354
00355     /* Sedimentation... */
00356     if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0)
00357         module_sedi(&ctl, met0, met1, atm, dt);
00358
00359     /* Isosurface... */
00360     if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00361         module_isosurf(&ctl, met0, met1, atm, cache);
00362
00363     /* Check final positions... */
00364     module_position(met0, met1, atm, dt);
00365
00366     /* Interpolate meteorological data... */
00367     if (ctl.met_dt_out > 0

```

```

00368         && (ctl.met_dt_out < ctl.dt_mod || fmod(t, ctl.met_dt_out) == 0))
00369         module_meteo(&ctl, met0, met1, atm);
00370
00371         /* Decay of particle mass... */
00372         if (ctl.tdec_trop > 0 && ctl.tdec_strat > 0)
00373             module_decay(&ctl, atm, dt);
00374
00375         /* OH chemistry... */
00376         if (ctl.oh_chem[0] > 0 && ctl.oh_chem[2] > 0)
00377             module_oh_chem(&ctl, met0, met1, atm, dt);
00378
00379         /* Dry deposition... */
00380         if (ctl.dry_depo[0] > 0)
00381             module_dry_deposition(&ctl, met0, met1, atm, dt);
00382
00383         /* Wet deposition... */
00384         if ((ctl.wet_depo[0] > 0 || ctl.wet_depo[2] > 0)
00385             && (ctl.wet_depo[4] > 0 || ctl.wet_depo[6] > 0))
00386             module_wet_deposition(&ctl, met0, met1, atm, dt);
00387
00388         /* Write output... */
00389         write_output(dirname, &ctl, met0, met1, atm, t);
00390     }
00391
00392     /* -----
00393     Finalize model run...
00394     ----- */
00395
00396     /* Report problem size... */
00397     printf("SIZE_NP = %d\n", atm->np);
00398     printf("SIZE_MPI_TASKS = %d\n", size);
00399     printf("SIZE_OMP_THREADS = %d\n", omp_get_max_threads());
00400     printf("SIZE_ACC_DEVICES = %d\n", num_devices);
00401
00402     /* Report memory usage... */
00403     printf("MEMORY_ATM = %g MByte\n", sizeof(atm_t) / 1024. / 1024.);
00404     printf("MEMORY_CACHE = %g MByte\n", sizeof(cache_t) / 1024. / 1024.);
00405     printf("MEMORY_METEO = %g MByte\n", 2 * sizeof(met_t) / 1024. / 1024.);
00406     printf("MEMORY_DYNAMIC = %g MByte\n", (sizeof(met_t)
00407         + 4 * NP * sizeof(double)
00408         + EX * EY * EP * sizeof(float)) /
00409         1024. / 1024.);
00410     printf("MEMORY_STATIC = %g MByte\n", (EX * EY * sizeof(double)
00411         + EX * EY * EP * sizeof(float)
00412         + 4 * GX * GY * GZ * sizeof(double)
00413         + 2 * GX * GY * GZ * sizeof(int)
00414         + 2 * GX * GY * sizeof(double)
00415         + GX * GY * sizeof(int)) / 1024. /
00416         1024.);
00417
00418     /* Delete data region on GPUs... */
00419     #ifdef _OPENACC
00420         SELECT_TIMER("DELETE_DATA_REGION", NVTX_GPU);
00421         #pragma acc exit data delete(ctl, atm, cache, met0, met1, dt, rs)
00422     #endif
00423
00424     /* Free... */
00425     SELECT_TIMER("FREE", NVTX_CPU);
00426     free(atm);
00427     free(cache);
00428     free(met0);
00429     free(met1);
00430     free(dt);
00431     free(rs);
00432
00433     /* Report timers... */
00434     PRINT_TIMERS;
00435 }
00436
00437 /* Finalize MPI... */
00438 #ifdef MPI
00439     MPI_Finalize();
00440 #endif
00441
00442 /* Stop timers... */
00443 STOP_TIMERS;
00444
00445 return EXIT_SUCCESS;
00446 }
00447
00448 /*****
00449
00450 void module_advection(
00451     met_t * met0,
00452     met_t * met1,
00453     atm_t * atm,
00454     double *dt) {

```

```

00455
00456 /* Set timer... */
00457 SELECT_TIMER("MODULE_ADVECTION", NVTX_GPU);
00458
00459 #ifdef _OPENACC
00460 #pragma acc data present(met0,met1,atm,dt)
00461 #pragma acc parallel loop independent gang vector
00462 #else
00463 #pragma omp parallel for default(shared)
00464 #endif
00465 for (int ip = 0; ip < atm->np; ip++)
00466     if (dt[ip] != 0) {
00467
00468         double u, v, w;
00469
00470         /* Interpolate meteorological data... */
00471         INTPOL_INIT;
00472         INTPOL_3D(u, 1);
00473         INTPOL_3D(v, 0);
00474         INTPOL_3D(w, 0);
00475
00476         /* Get position of the mid point... */
00477         double dtm = atm->time[ip] + 0.5 * dt[ip];
00478         double xm0 =
00479             atm->lon[ip] + DX2DEG(0.5 * dt[ip] * u / 1000., atm->lat[ip]);
00480         double xm1 = atm->lat[ip] + DY2DEG(0.5 * dt[ip] * v / 1000.);
00481         double xm2 = atm->p[ip] + 0.5 * dt[ip] * w;
00482
00483         /* Interpolate meteorological data for mid point... */
00484         intpol_met_time_3d(met0, met0->u, met1, met1->u,
00485             dtm, xm2, xm0, xm1, &u, ci, cw, 1);
00486         intpol_met_time_3d(met0, met0->v, met1, met1->v,
00487             dtm, xm2, xm0, xm1, &v, ci, cw, 0);
00488         intpol_met_time_3d(met0, met0->w, met1, met1->w,
00489             dtm, xm2, xm0, xm1, &w, ci, cw, 0);
00490
00491         /* Save new position... */
00492         atm->time[ip] += dt[ip];
00493         atm->lon[ip] += DX2DEG(dt[ip] * u / 1000., xm1);
00494         atm->lat[ip] += DY2DEG(dt[ip] * v / 1000.);
00495         atm->p[ip] += dt[ip] * w;
00496     }
00497 }
00498
00499 /*****
00500
00501 void module_convection(
00502     ctl_t * ctl,
00503     met_t * met0,
00504     met_t * met1,
00505     atm_t * atm,
00506     double *dt,
00507     double *rs) {
00508
00509     /* Set timer... */
00510     SELECT_TIMER("MODULE_CONVECTION", NVTX_GPU);
00511
00512     #ifdef _OPENACC
00513     #pragma acc data present(ctl,met0,met1,atm,dt,rs)
00514     #pragma acc parallel loop independent gang vector
00515     #else
00516     #pragma omp parallel for default(shared)
00517     #endif
00518     for (int ip = 0; ip < atm->np; ip++)
00519         if (dt[ip] != 0) {
00520
00521             double cape, pel, ps;
00522
00523             /* Interpolate CAPE... */
00524             INTPOL_INIT;
00525             INTPOL_2D(cape, 1);
00526
00527             /* Check threshold... */
00528             if (isfinite(cape) && cape >= ctl->conv_cape) {
00529
00530                 /* Interpolate equilibrium level... */
00531                 INTPOL_2D(pel, 0);
00532
00533                 /* Check whether particle is above cloud top... */
00534                 if (!isfinite(pel) || atm->p[ip] < pel)
00535                     continue;
00536
00537                 /* Interpolate surface pressure... */
00538                 INTPOL_2D(ps, 0);
00539
00540                 /* Redistribute particle in cloud column... */
00541                 atm->p[ip] = ps + (pel - ps) * rs[ip];

```

```

00542     }
00543     }
00544 }
00545
00546 /*****
00547
00548 void module_decay(
00549     ctl_t * ctl,
00550     atm_t * atm,
00551     double *dt) {
00552
00553     /* Set timer... */
00554     SELECT_TIMER("MODULE_DECAY", NVTX_GPU);
00555
00556     /* Check quantity flags... */
00557     if (ctl->qnt_m < 0)
00558         ERRMSG("Module needs quantity mass!");
00559
00560 #ifdef _OPENACC
00561 #pragma acc data present(ctl,atm,dt)
00562 #pragma acc parallel loop independent gang vector
00563 #else
00564 #pragma omp parallel for default(shared)
00565 #endif
00566     for (int ip = 0; ip < atm->np; ip++)
00567         if (dt[ip] != 0) {
00568
00569             /* Get tropopause pressure... */
00570             double pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00571
00572             /* Get weighting factor... */
00573             double w;
00574             double p1 = pt * 0.866877899;
00575             double p0 = pt / 0.866877899;
00576             if (atm->p[ip] > p0)
00577                 w = 1;
00578             else if (atm->p[ip] < p1)
00579                 w = 0;
00580             else
00581                 w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00582
00583             /* Set lifetime... */
00584             double tdec = w * ctl->tdec_trop + (1 - w) * ctl->tdec_strat;
00585
00586             /* Calculate exponential decay... */
00587             atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] / tdec);
00588         }
00589 }
00590
00591 /*****
00592
00593 void module_diffusion_meso(
00594     ctl_t * ctl,
00595     met_t * met0,
00596     met_t * met1,
00597     atm_t * atm,
00598     cache_t * cache,
00599     double *dt,
00600     double *rs) {
00601
00602     /* Set timer... */
00603     SELECT_TIMER("MODULE_TURBMESO", NVTX_GPU);
00604
00605 #ifdef _OPENACC
00606 #pragma acc data present(ctl,met0,met1,atm,cache,dt,rs)
00607 #pragma acc parallel loop independent gang vector
00608 #else
00609 #pragma omp parallel for default(shared)
00610 #endif
00611     for (int ip = 0; ip < atm->np; ip++)
00612         if (dt[ip] != 0) {
00613
00614             double u[16], v[16], w[16];
00615
00616             /* Get indices... */
00617             int ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
00618             int iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
00619             int iz = locate_irr(met0->p, met0->np, atm->p[ip]);
00620
00621             /* Caching of wind standard deviations... */
00622             if (cache->tsig[ix][iy][iz] != met0->time) {
00623
00624                 /* Collect local wind data... */
00625                 u[0] = met0->u[ix][iy][iz];
00626                 u[1] = met0->u[ix + 1][iy][iz];
00627                 u[2] = met0->u[ix][iy + 1][iz];
00628                 u[3] = met0->u[ix + 1][iy + 1][iz];

```

```

00629     u[4] = met0->u[ix][iy][iz + 1];
00630     u[5] = met0->u[ix + 1][iy][iz + 1];
00631     u[6] = met0->u[ix][iy + 1][iz + 1];
00632     u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00633
00634     v[0] = met0->v[ix][iy][iz];
00635     v[1] = met0->v[ix + 1][iy][iz];
00636     v[2] = met0->v[ix][iy + 1][iz];
00637     v[3] = met0->v[ix + 1][iy + 1][iz];
00638     v[4] = met0->v[ix][iy][iz + 1];
00639     v[5] = met0->v[ix + 1][iy][iz + 1];
00640     v[6] = met0->v[ix][iy + 1][iz + 1];
00641     v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00642
00643     w[0] = met0->w[ix][iy][iz];
00644     w[1] = met0->w[ix + 1][iy][iz];
00645     w[2] = met0->w[ix][iy + 1][iz];
00646     w[3] = met0->w[ix + 1][iy + 1][iz];
00647     w[4] = met0->w[ix][iy][iz + 1];
00648     w[5] = met0->w[ix + 1][iy][iz + 1];
00649     w[6] = met0->w[ix][iy + 1][iz + 1];
00650     w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00651
00652     /* Collect local wind data... */
00653     u[8] = met1->u[ix][iy][iz];
00654     u[9] = met1->u[ix + 1][iy][iz];
00655     u[10] = met1->u[ix][iy + 1][iz];
00656     u[11] = met1->u[ix + 1][iy + 1][iz];
00657     u[12] = met1->u[ix][iy][iz + 1];
00658     u[13] = met1->u[ix + 1][iy][iz + 1];
00659     u[14] = met1->u[ix][iy + 1][iz + 1];
00660     u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00661
00662     v[8] = met1->v[ix][iy][iz];
00663     v[9] = met1->v[ix + 1][iy][iz];
00664     v[10] = met1->v[ix][iy + 1][iz];
00665     v[11] = met1->v[ix + 1][iy + 1][iz];
00666     v[12] = met1->v[ix][iy][iz + 1];
00667     v[13] = met1->v[ix + 1][iy][iz + 1];
00668     v[14] = met1->v[ix][iy + 1][iz + 1];
00669     v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00670
00671     w[8] = met1->w[ix][iy][iz];
00672     w[9] = met1->w[ix + 1][iy][iz];
00673     w[10] = met1->w[ix][iy + 1][iz];
00674     w[11] = met1->w[ix + 1][iy + 1][iz];
00675     w[12] = met1->w[ix][iy][iz + 1];
00676     w[13] = met1->w[ix + 1][iy][iz + 1];
00677     w[14] = met1->w[ix][iy + 1][iz + 1];
00678     w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00679
00680     /* Get standard deviations of local wind data... */
00681     cache->usig[ix][iy][iz] = (float) stddev(u, 16);
00682     cache->vsig[ix][iy][iz] = (float) stddev(v, 16);
00683     cache->wsig[ix][iy][iz] = (float) stddev(w, 16);
00684     cache->tsig[ix][iy][iz] = met0->time;
00685 }
00686
00687 /* Set temporal correlations for mesoscale fluctuations... */
00688 double r = 1 - 2 * fabs(dt[ip]) / ctl->dt_met;
00689 double r2 = sqrt(1 - r * r);
00690
00691 /* Calculate horizontal mesoscale wind fluctuations... */
00692 if (ctl->turb_mesox > 0) {
00693     cache->up[ip] = (float)
00694         (r * cache->up[ip]
00695          + r2 * rs[3 * ip] * ctl->turb_mesox * cache->usig[ix][iy][iz]);
00696     atm->lon[ip] += DX2DEG(cache->up[ip] * dt[ip] / 1000., atm->lat[ip]);
00697
00698     cache->vp[ip] = (float)
00699         (r * cache->vp[ip]
00700          + r2 * rs[3 * ip + 1] * ctl->turb_mesox * cache->vsig[ix][iy][iz]);
00701     atm->lat[ip] += DY2DEG(cache->vp[ip] * dt[ip] / 1000.);
00702 }
00703
00704 /* Calculate vertical mesoscale wind fluctuations... */
00705 if (ctl->turb_mesoz > 0) {
00706     cache->wp[ip] = (float)
00707         (r * cache->wp[ip]
00708          + r2 * rs[3 * ip + 2] * ctl->turb_mesoz * cache->wsig[ix][iy][iz]);
00709     atm->p[ip] += cache->wp[ip] * dt[ip];
00710 }
00711 }
00712 }
00713
00714 /*****
00715

```



```

00716 void module_diffusion_turb(
00717     ctl_t * ctl,
00718     atm_t * atm,
00719     double *dt,
00720     double *rs) {
00721
00722     /* Set timer... */
00723     SELECT_TIMER("MODULE_TURBDIFF", NVTX_GPU);
00724
00725 #ifdef _OPENACC
00726 #pragma acc data present(ctl,atm,dt,rs)
00727 #pragma acc parallel loop independent gang vector
00728 #else
00729 #pragma omp parallel for default(shared)
00730 #endif
00731     for (int ip = 0; ip < atm->np; ip++)
00732         if (dt[ip] != 0) {
00733
00734             /* Get tropopause pressure... */
00735             double pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00736
00737             /* Get weighting factor... */
00738             double w;
00739             double p1 = pt * 0.866877899;
00740             double p0 = pt / 0.866877899;
00741             if (atm->p[ip] > p0)
00742                 w = 1;
00743             else if (atm->p[ip] < p1)
00744                 w = 0;
00745             else
00746                 w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00747
00748             /* Set diffusivity... */
00749             double dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00750             double dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00751
00752             /* Horizontal turbulent diffusion... */
00753             if (dx > 0) {
00754                 double sigma = sqrt(2.0 * dx * fabs(dt[ip]));
00755                 atm->lon[ip] += DX2DEG(rs[3 * ip] * sigma / 1000., atm->lat[ip]);
00756                 atm->lat[ip] += DY2DEG(rs[3 * ip + 1] * sigma / 1000.);
00757             }
00758
00759             /* Vertical turbulent diffusion... */
00760             if (dz > 0) {
00761                 double sigma = sqrt(2.0 * dz * fabs(dt[ip]));
00762                 atm->p[ip]
00763                     += DZ2DP(rs[3 * ip + 2] * sigma / 1000., atm->p[ip]);
00764             }
00765         }
00766 }
00767
00768 /*****
00769
00770 void module_dry_deposition(
00771     ctl_t * ctl,
00772     met_t * met0,
00773     met_t * met1,
00774     atm_t * atm,
00775     double *dt) {
00776
00777     /* Set timer... */
00778     SELECT_TIMER("MODULE_DRYDEPO", NVTX_GPU);
00779
00780     /* Width of the surface layer [hPa]. */
00781     const double dp = 30.;
00782
00783     /* Check quantity flags... */
00784     if (ctl->qnt_m < 0)
00785         ERRMSG("Module needs quantity mass!");
00786
00787 #ifdef _OPENACC
00788 #pragma acc data present(ctl,met0,met1,atm,dt)
00789 #pragma acc parallel loop independent gang vector
00790 #else
00791 #pragma omp parallel for default(shared)
00792 #endif
00793     for (int ip = 0; ip < atm->np; ip++)
00794         if (dt[ip] != 0) {
00795
00796             double ps, t, v_dep;
00797
00798             /* Get surface pressure... */
00799             INTPOL_INIT;
00800             INTPOL_2D(ps, 1);
00801
00802             /* Check whether particle is above the surface layer... */

```

```

00803     if (atm->p[ip] < ps - dp)
00804         continue;
00805
00806     /* Set width of surface layer... */
00807     double dz = 1000. * (Z(ps - dp) - Z(ps));
00808
00809     /* Calculate sedimentation velocity for particles... */
00810     if (ctl->qnt_r >= 0 && ctl->qnt_rho >= 0) {
00811
00812         /* Get temperature... */
00813         INTPOL_3D(t, 1);
00814
00815         /* Set deposition velocity... */
00816         v_dep = sedi(atm->p[ip], t, atm->q[ctl->qnt_r][ip],
00817                     atm->q[ctl->qnt_rho][ip]);
00818     }
00819
00820     /* Use explicit sedimentation velocity for gases... */
00821     else
00822         v_dep = ctl->dry_depo[0];
00823
00824     /* Calculate loss of mass based on deposition velocity... */
00825     atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] * v_dep / dz);
00826 }
00827 }
00828
00829 /*****
00830
00831 void module_isosurf_init(
00832     ctl_t * ctl,
00833     met_t * met0,
00834     met_t * met1,
00835     atm_t * atm,
00836     cache_t * cache) {
00837
00838     FILE *in;
00839
00840     char line[LEN];
00841
00842     double t;
00843
00844     /* Set timer... */
00845     SELECT_TIMER("MODULE_ISOSURF", NVTX_GPU);
00846
00847     /* Init... */
00848     INTPOL_INIT;
00849
00850     /* Save pressure... */
00851     if (ctl->isosurf == 1)
00852         for (int ip = 0; ip < atm->np; ip++)
00853             cache->iso_var[ip] = atm->p[ip];
00854
00855     /* Save density... */
00856     else if (ctl->isosurf == 2)
00857         for (int ip = 0; ip < atm->np; ip++) {
00858             INTPOL_3D(t, 1);
00859             cache->iso_var[ip] = atm->p[ip] / t;
00860         }
00861
00862     /* Save potential temperature... */
00863     else if (ctl->isosurf == 3)
00864         for (int ip = 0; ip < atm->np; ip++) {
00865             INTPOL_3D(t, 1);
00866             cache->iso_var[ip] = THETA(atm->p[ip], t);
00867         }
00868
00869     /* Read balloon pressure data... */
00870     else if (ctl->isosurf == 4) {
00871
00872         /* Write info... */
00873         printf("Read balloon pressure data: %s\n", ctl->balloon);
00874
00875         /* Open file... */
00876         if (!(in = fopen(ctl->balloon, "r")))
00877             ERRMSG("Cannot open file!");
00878
00879         /* Read pressure time series... */
00880         while (fgets(line, LEN, in))
00881             if (sscanf(line, "%lg %lg", &(cache->iso_ts[cache->iso_n]),
00882                     &(cache->iso_ps[cache->iso_n])) == 2)
00883                 if (++cache->iso_n > NP)
00884                     ERRMSG("Too many data points!");
00885
00886         /* Check number of points... */
00887         if (cache->iso_n < 1)
00888             ERRMSG("Could not read any data!");
00889

```

```

00890     /* Close file... */
00891     fclose(in);
00892 }
00893 }
00894
00895 /*****
00896 void module_isosurf(
00897     ctl_t * ctl,
00898     met_t * met0,
00899     met_t * met1,
00900     atm_t * atm,
00901     cache_t * cache) {
00902
00903     /* Set timer... */
00904     SELECT_TIMER("MODULE_ISOSURF", NVTX_GPU);
00905
00906 #ifdef _OPENACC
00907 #pragma acc data present(ctl,met0,met1,atm,cache)
00908 #pragma acc parallel loop independent gang vector
00909 #else
00910 #pragma omp parallel for default(shared)
00911 #endif
00912     for (int ip = 0; ip < atm->np; ip++) {
00913
00914         double t;
00915
00916         /* Init... */
00917         INTPOL_INIT;
00918
00919         /* Restore pressure... */
00920         if (ctl->isosurf == 1)
00921             atm->p[ip] = cache->iso_var[ip];
00922
00923         /* Restore density... */
00924         else if (ctl->isosurf == 2) {
00925             INTPOL_3D(t, 1);
00926             atm->p[ip] = cache->iso_var[ip] * t;
00927         }
00928
00929         /* Restore potential temperature... */
00930         else if (ctl->isosurf == 3) {
00931             INTPOL_3D(t, 1);
00932             atm->p[ip] = 1000. * pow(cache->iso_var[ip] / t, -1. / 0.286);
00933         }
00934
00935         /* Interpolate pressure... */
00936         else if (ctl->isosurf == 4) {
00937             if (atm->time[ip] <= cache->iso_ts[0])
00938                 atm->p[ip] = cache->iso_ps[0];
00939             else if (atm->time[ip] >= cache->iso_ts[cache->iso_n - 1])
00940                 atm->p[ip] = cache->iso_ps[cache->iso_n - 1];
00941             else {
00942                 int idx = locate_irr(cache->iso_ts, cache->iso_n, atm->time[ip]);
00943                 atm->p[ip] = LIN(cache->iso_ts[idx], cache->iso_ps[idx],
00944                             cache->iso_ts[idx + 1], cache->iso_ps[idx + 1],
00945                             atm->time[ip]);
00946             }
00947         }
00948     }
00949 }
00950 }
00951
00952 /*****
00953 void module_meteo(
00954     ctl_t * ctl,
00955     met_t * met0,
00956     met_t * met1,
00957     atm_t * atm) {
00958
00959     /* Set timer... */
00960     SELECT_TIMER("MODULE_METEO", NVTX_GPU);
00961
00962     /* Check quantity flags... */
00963     if (ctl->qnt_tsts >= 0)
00964         if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
00965             ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
00966
00967 #ifdef _OPENACC
00968 #pragma acc data present(ctl,met0,met1,atm)
00969 #pragma acc parallel loop independent gang vector
00970 #else
00971 #pragma omp parallel for default(shared)
00972 #endif
00973     for (int ip = 0; ip < atm->np; ip++) {
00974
00975         double ps, ts, zs, us, vs, pt, pc, cl, plcl, plfc, pel, cape, pv, t, tt,

```

```

00977     u, v, w, h2o, h2ot, o3, lwc, iwc, z, zt;
00978
00979     /* Interpolate meteorological data... */
00980     INTPOL_INIT;
00981     INTPOL_TIME_ALL(atm->time[ip], atm->p[ip], atm->lon[ip], atm->lat[ip]);
00982
00983     /* Set quantities... */
00984     ATM_SET(qnt_ps, ps);
00985     ATM_SET(qnt_ts, ts);
00986     ATM_SET(qnt_zs, zs);
00987     ATM_SET(qnt_us, us);
00988     ATM_SET(qnt_vs, vs);
00989     ATM_SET(qnt_pt, pt);
00990     ATM_SET(qnt_tt, tt);
00991     ATM_SET(qnt_zt, zt);
00992     ATM_SET(qnt_h2ot, h2ot);
00993     ATM_SET(qnt_p, atm->p[ip]);
00994     ATM_SET(qnt_z, z);
00995     ATM_SET(qnt_t, t);
00996     ATM_SET(qnt_u, u);
00997     ATM_SET(qnt_v, v);
00998     ATM_SET(qnt_w, w);
00999     ATM_SET(qnt_h2o, h2o);
01000     ATM_SET(qnt_o3, o3);
01001     ATM_SET(qnt_lwc, lwc);
01002     ATM_SET(qnt_iwc, iwc);
01003     ATM_SET(qnt_pc, pc);
01004     ATM_SET(qnt_cl, cl);
01005     ATM_SET(qnt_plcl, plcl);
01006     ATM_SET(qnt_plfc, plfc);
01007     ATM_SET(qnt_pel, pel);
01008     ATM_SET(qnt_cape, cape);
01009     ATM_SET(qnt_hno3, clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip]));
01010     ATM_SET(qnt_oh, clim_oh(atm->time[ip], atm->lat[ip], atm->p[ip]));
01011     ATM_SET(qnt_vh, sqrt(u * u + v * v));
01012     ATM_SET(qnt_vz, -1e3 * H0 / atm->p[ip] * w);
01013     ATM_SET(qnt_psat, PSAT(t));
01014     ATM_SET(qnt_psice, PSICE(t));
01015     ATM_SET(qnt_pw, PW(atm->p[ip], h2o));
01016     ATM_SET(qnt_sh, SH(h2o));
01017     ATM_SET(qnt_rh, RH(atm->p[ip], t, h2o));
01018     ATM_SET(qnt_rhice, RHICE(atm->p[ip], t, h2o));
01019     ATM_SET(qnt_theta, THETA(atm->p[ip], t));
01020     ATM_SET(qnt_tvirt, TVIRT(t, h2o));
01021     ATM_SET(qnt_lapse, lapse_rate(t, h2o));
01022     ATM_SET(qnt_pv, pv);
01023     ATM_SET(qnt_tdew, TDEW(atm->p[ip], h2o));
01024     ATM_SET(qnt_tice, TICE(atm->p[ip], h2o));
01025     ATM_SET(qnt_tnat,
01026             nat_temperature(atm->p[ip], h2o,
01027                             clim_hno3(atm->time[ip], atm->lat[ip],
01028                                         atm->p[ip]) * 1e-9));
01029     ATM_SET(qnt_tsts,
01030             0.5 * (atm->q[ctl->qnt_tice][ip] + atm->q[ctl->qnt_tnat][ip]));
01031 }
01032 }
01033
01034 /*****
01035 void module_position(
01036     met_t * met0,
01037     met_t * met1,
01038     atm_t * atm,
01039     double *dt) {
01040
01041     /* Set timer... */
01042     SELECT_TIMER("MODULE_POSITION", NVTX_GPU);
01043
01044     #ifdef _OPENACC
01045     #pragma acc data present(met0,met1,atm,dt)
01046     #pragma acc parallel loop independent gang vector
01047     #else
01048     #pragma omp parallel for default(shared)
01049     #endif
01050     for (int ip = 0; ip < atm->np; ip++)
01051         if (dt[ip] != 0) {
01052
01053             /* Calculate modulo... */
01054             atm->lon[ip] = FMOD(atm->lon[ip], 360.);
01055             atm->lat[ip] = FMOD(atm->lat[ip], 360.);
01056
01057             /* Check latitude... */
01058             while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
01059                 if (atm->lat[ip] > 90) {
01060                     atm->lat[ip] = 180 - atm->lat[ip];
01061                     atm->lon[ip] += 180;
01062                 }
01063             }

```

```

01064         if (atm->lat[ip] < -90) {
01065             atm->lat[ip] = -180 - atm->lat[ip];
01066             atm->lon[ip] += 180;
01067         }
01068     }
01069
01070     /* Check longitude... */
01071     while (atm->lon[ip] < -180)
01072         atm->lon[ip] += 360;
01073     while (atm->lon[ip] >= 180)
01074         atm->lon[ip] -= 360;
01075
01076     /* Check pressure... */
01077     if (atm->p[ip] < met0->p[met0->np - 1])
01078         atm->p[ip] = met0->p[met0->np - 1];
01079     else if (atm->p[ip] > 300.) {
01080         double ps;
01081         INTPOL_INIT;
01082         INTPOL_2D(ps, 1);
01083         if (atm->p[ip] > ps)
01084             atm->p[ip] = ps;
01085     }
01086 }
01087 }
01088
01089 /*****
01090
01091 void module_rng_init(
01092     void) {
01093
01094     /* Set timer... */
01095     SELECT_TIMER("MODULE_RNG", NVTX_GPU);
01096
01097     /* Initialize random number generator... */
01098     #ifdef _OPENACC
01099
01100     if (curandCreateGenerator(&rng, CURAND_RNG_PSEUDO_DEFAULT)
01101         != CURAND_STATUS_SUCCESS)
01102         ERRMSG("Cannot create random number generator!");
01103     if (curandSetStream(rng, (cudaStream_t) acc_get_cuda_stream(acc_async_sync))
01104         != CURAND_STATUS_SUCCESS)
01105         ERRMSG("Cannot set stream for random number generator!");
01106
01107     #else
01108
01109     gsl_rng_env_setup();
01110     if (omp_get_max_threads() > NTHREADS)
01111         ERRMSG("Too many threads!");
01112     for (int i = 0; i < NTHREADS; i++) {
01113         rng[i] = gsl_rng_alloc(gsl_rng_default);
01114         gsl_rng_set(rng[i], gsl_rng_default_seed + (long unsigned) i);
01115     }
01116
01117     #endif
01118 }
01119
01120 /*****
01121
01122 void module_rng(
01123     double *rs,
01124     size_t n,
01125     int method) {
01126
01127     /* Set timer... */
01128     SELECT_TIMER("MODULE_RNG", NVTX_GPU);
01129
01130     #ifdef _OPENACC
01131
01132     #pragma acc host_data use_device(rs)
01133     {
01134         /* Uniform distribution... */
01135         if (method == 0) {
01136             if (curandGenerateUniform(rng, rs, n)
01137                 != CURAND_STATUS_SUCCESS)
01138                 ERRMSG("Cannot create random numbers!");
01139         }
01140
01141         /* Normal distribution... */
01142         else if (method == 1) {
01143             if (curandGenerateNormalDouble(rng, rs, n, 0.0, 1.0)
01144                 != CURAND_STATUS_SUCCESS)
01145                 ERRMSG("Cannot create random numbers!");
01146         }
01147     }
01148
01149     #else
01150

```

```

01151  /* Uniform distribution... */
01152  if (method == 0) {
01153  #pragma omp parallel for default(shared)
01154      for (size_t i = 0; i < n; ++i)
01155          rs[i] = gsl_rng_uniform(rng[omp_get_thread_num()]);
01156  }
01157
01158  /* Normal distribution... */
01159  else if (method == 1) {
01160  #pragma omp parallel for default(shared)
01161      for (size_t i = 0; i < n; ++i)
01162          rs[i] = gsl_rng_gaussian_ziggurat(rng[omp_get_thread_num()], 1.0);
01163  }
01164  #endif
01165
01166  }
01167
01168  /*****
01169
01170  void module_sedi(
01171      ctl_t * ctl,
01172      met_t * met0,
01173      met_t * met1,
01174      atm_t * atm,
01175      double *dt) {
01176
01177      /* Set timer... */
01178      SELECT_TIMER("MODULE_SEDI", NVTX_GPU);
01179
01180  #ifdef _OPENACC
01181  #pragma acc data present(ctl,met0,met1,atm,dt)
01182  #pragma acc parallel loop independent gang vector
01183  #else
01184  #pragma omp parallel for default(shared)
01185  #endif
01186      for (int ip = 0; ip < atm->np; ip++)
01187          if (dt[ip] != 0) {
01188
01189              /* Get temperature... */
01190              double t;
01191              INTPOL_INIT;
01192              INTPOL_3D(t, 1);
01193
01194              /* Sedimentation velocity... */
01195              double v_s = sedi(atm->p[ip], t, atm->q[ctl->qnt_r][ip],
01196                              atm->q[ctl->qnt_rho][ip]);
01197
01198              /* Calculate pressure change... */
01199              atm->p[ip] += DZ2DP(v_s * dt[ip] / 1000., atm->p[ip]);
01200          }
01201  }
01202
01203  /*****
01204
01205  void module_oh_chem(
01206      ctl_t * ctl,
01207      met_t * met0,
01208      met_t * met1,
01209      atm_t * atm,
01210      double *dt) {
01211
01212      /* Set timer... */
01213      SELECT_TIMER("MODULE_OHCHEM", NVTX_GPU);
01214
01215      /* Check quantity flags... */
01216      if (ctl->qnt_m < 0)
01217          ERRMSG("Module needs quantity mass!");
01218
01219  #ifdef _OPENACC
01220  #pragma acc data present(ctl,met0,met1,atm,dt)
01221  #pragma acc parallel loop independent gang vector
01222  #else
01223  #pragma omp parallel for default(shared)
01224  #endif
01225      for (int ip = 0; ip < atm->np; ip++)
01226          if (dt[ip] != 0) {
01227
01228              /* Get temperature... */
01229              double t;
01230              INTPOL_INIT;
01231              INTPOL_3D(t, 1);
01232
01233              /* Calculate molecular density... */
01234              double M = 7.243e21 * (atm->p[ip] / P0) / t;
01235
01236              /* Calculate rate coefficient for X + OH + M -> XOH + M
01237              (JPL Publication 15-10) ... */

```

```

01238     double k0 = ctl->oh_chem[0] *
01239         (ctl->oh_chem[1] > 0 ? pow(t / 300., -ctl->oh_chem[1]) : 1.);
01240     double ki = ctl->oh_chem[2] *
01241         (ctl->oh_chem[3] > 0 ? pow(t / 300., -ctl->oh_chem[3]) : 1.);
01242     double c = log10(k0 * M / ki);
01243     double k = k0 * M / (1. + k0 * M / ki) * pow(0.6, 1. / (1. + c * c));
01244
01245     /* Calculate exponential decay... */
01246     atm->q[ctl->qnt_m][ip] *=
01247         exp(-dt[ip] * k * clim_oh(atm->time[ip], atm->lat[ip], atm->p[ip]));
01248 }
01249 }
01250
01251 /*****
01252
01253 void module_wet_deposition(
01254     ctl_t * ctl,
01255     met_t * met0,
01256     met_t * met1,
01257     atm_t * atm,
01258     double *dt) {
01259
01260     /* Set timer... */
01261     SELECT_TIMER("MODULE_WETDEPO", NVTX_GPU);
01262
01263     /* Check quantity flags... */
01264     if (ctl->qnt_m < 0)
01265         ERRMSG("Module needs quantity mass!");
01266
01267 #ifdef _OPENACC
01268 #pragma acc data present(ctl,met0,met1,atm,dt)
01269 #pragma acc parallel loop independent gang vector
01270 #else
01271 #pragma omp parallel for default(shared)
01272 #endif
01273     for (int ip = 0; ip < atm->np; ip++)
01274         if (dt[ip] != 0) {
01275
01276             double cl, dz, h, lambda = 0, t, iwc, lwc, pc;
01277
01278             int inside;
01279
01280             /* Check whether particle is below cloud top... */
01281             INTPOL_INIT;
01282             INTPOL_2D(pc, 1);
01283             if (!isfinite(pc) || atm->p[ip] <= pc)
01284                 continue;
01285
01286             /* Estimate precipitation rate (Pisso et al., 2019)... */
01287             INTPOL_2D(cl, 0);
01288             double Is = pow(2. * cl, 1. / 0.36);
01289             if (Is < 0.01)
01290                 continue;
01291
01292             /* Check whether particle is inside or below cloud... */
01293             INTPOL_3D(lwc, 1);
01294             INTPOL_3D(iwc, 0);
01295             inside = (iwc > 0 || lwc > 0);
01296
01297             /* Calculate in-cloud scavenging coefficient... */
01298             if (inside) {
01299
01300                 /* Use exponential dependency for particles... */
01301                 if (ctl->wet_depo[0] > 0)
01302                     lambda = ctl->wet_depo[0] * pow(Is, ctl->wet_depo[1]);
01303
01304                 /* Use Henry's law for gases... */
01305                 else if (ctl->wet_depo[2] > 0) {
01306
01307                     /* Get temperature... */
01308                     INTPOL_3D(t, 0);
01309
01310                     /* Get Henry's constant (Sander, 2015)... */
01311                     h = ctl->wet_depo[2]
01312                         * exp(ctl->wet_depo[3] * (1. / t - 1. / 298.15));
01313
01314                     /* Estimate depth of cloud layer... */
01315                     dz = 1e3 * Z(pc);
01316
01317                     /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
01318                     lambda = h * RI * t * Is / 3.6e6 * dz;
01319                 }
01320             }
01321
01322             /* Calculate below-cloud scavenging coefficient... */
01323             else {
01324

```

```

01325      /* Use exponential dependency for particles... */
01326      if (ctl->wet_depo[4] > 0)
01327          lambda = ctl->wet_depo[4] * pow(Is, ctl->wet_depo[5]);
01328
01329      /* Use Henry's law for gases... */
01330      else if (ctl->wet_depo[6] > 0) {
01331
01332          /* Get temperature... */
01333          INTPOL_3D(t, 0);
01334
01335          /* Get Henry's constant (Sander, 2015)... */
01336          h = ctl->wet_depo[6]
01337              * exp(ctl->wet_depo[7] * (1. / t - 1. / 298.15));
01338
01339          /* Estimate depth of cloud layer... */
01340          dz = 1e3 * Z(pc);
01341
01342          /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
01343          lambda = h * RI * t * Is / 3.6e6 * dz;
01344      }
01345  }
01346
01347      /* Calculate exponential decay of mass... */
01348      atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] * lambda);
01349  }
01350 }
01351
01352 /*****
01353 void write_output(
01354     const char *dirname,
01355     ctl_t * ctl,
01356     met_t * met0,
01357     met_t * met1,
01358     atm_t * atm,
01359     double t) {
01360
01361     char filename[2 * LEN];
01362
01363     double r;
01364
01365     int year, mon, day, hour, min, sec, updated = 0;
01366
01367     /* Get time... */
01368     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01369
01370     /* Write atmospheric data... */
01371     if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01372         if (!updated) {
01373             #ifdef _OPENACC
01374                 SELECT_TIMER("UPDATE_HOST", NVTX_D2H);
01375             #pragma acc update host(atm[:1])
01376             #endif
01377             updated = 1;
01378         }
01379         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.tab",
01380             dirname, ctl->atm_basename, year, mon, day, hour, min);
01381         write_atm(filename, ctl, atm, t);
01382     }
01383 }
01384
01385     /* Write gridded data... */
01386     if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01387         if (!updated) {
01388             #ifdef _OPENACC
01389                 SELECT_TIMER("UPDATE_HOST", NVTX_D2H);
01390             #pragma acc update host(atm[:1])
01391             #endif
01392             updated = 1;
01393         }
01394         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01395             dirname, ctl->grid_basename, year, mon, day, hour, min);
01396         write_grid(filename, ctl, met0, met1, atm, t);
01397     }
01398
01399     /* Write CSI data... */
01400     if (ctl->csi_basename[0] != '-') {
01401         if (!updated) {
01402             #ifdef _OPENACC
01403                 SELECT_TIMER("UPDATE_HOST", NVTX_D2H);
01404             #pragma acc update host(atm[:1])
01405             #endif
01406             updated = 1;
01407         }
01408         sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01409         write_csi(filename, ctl, atm, t);
01410     }
01411 }

```



```

01412  /* Write ensemble data... */
01413  if (ctl->ens_basename[0] != '-') {
01414      if (!updated) {
01415 #ifdef _OPENACC
01416         SELECT_TIMER("UPDATE_HOST", NVTX_D2H);
01417 #pragma acc update host(atm[:1])
01418 #endif
01419         updated = 1;
01420     }
01421     sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01422     write_ens(filename, ctl, atm, t);
01423 }
01424
01425  /* Write profile data... */
01426  if (ctl->prof_basename[0] != '-') {
01427      if (!updated) {
01428 #ifdef _OPENACC
01429         SELECT_TIMER("UPDATE_HOST", NVTX_D2H);
01430 #pragma acc update host(atm[:1])
01431 #endif
01432         updated = 1;
01433     }
01434     sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01435     write_prof(filename, ctl, met0, met1, atm, t);
01436 }
01437
01438  /* Write sample data... */
01439  if (ctl->sample_basename[0] != '-') {
01440      if (!updated) {
01441 #ifdef _OPENACC
01442         SELECT_TIMER("UPDATE_HOST", NVTX_D2H);
01443 #pragma acc update host(atm[:1])
01444 #endif
01445         updated = 1;
01446     }
01447     sprintf(filename, "%s/%s.tab", dirname, ctl->sample_basename);
01448     write_sample(filename, ctl, met0, met1, atm, t);
01449 }
01450
01451  /* Write station data... */
01452  if (ctl->stat_basename[0] != '-') {
01453      if (!updated) {
01454 #ifdef _OPENACC
01455         SELECT_TIMER("UPDATE_HOST", NVTX_D2H);
01456 #pragma acc update host(atm[:1])
01457 #endif
01458         updated = 1;
01459     }
01460     sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01461     write_station(filename, ctl, atm, t);
01462 }
01463 }

```

## 5.39 tropo.c File Reference

### Functions

- void [add\\_text\\_attribute](#) (int ncid, char \*varname, char \*attrname, char \*text)
- int [main](#) (int argc, char \*argv[])

#### 5.39.1 Detailed Description

Create tropopause data set from meteorological data.

Definition in file [tropo.c](#).

#### 5.39.2 Function Documentation

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

Definition at line 337 of file [tropo.c](#).

```
00341     {
00342
00343     int varid;
00344
00345     NC(nc_inq_varid(ncid, varname, &varid));
00346     NC(nc_put_att_text(ncid, varid, attrname, strlen(text), text));
00347 }
```

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

Definition at line 41 of file [tropo.c](#).

```
00043     {
00044
00045     ctl_t ctl;
00046
00047     met_t *met;
00048
00049     static double pt[EX * EY], qt[EX * EY], zt[EX * EY], tt[EX * EY], lon, lon0,
00050     lon1, lons[EX], dlon, lat, lat0, lat1, lats[EY], dlat, cw[3];
00051
00052     static int init, i, ix, iy, nx, ny, nt, ncid, dims[3], timid, lonid, latid,
00053     clppid, clpqid, clptid, clpzyd, dynpid, dynqid, dyntid, dynzid, wmolpid,
00054     wmolqid, wmoltid, wmolzid, wmo2pid, wmo2qid, wmo2tid, wmo2zid, h2o, ci[3];
00055
00056     static size_t count[10], start[10];
00057
00058     /* Allocate... */
00059     ALLOC(met, met_t, 1);
00060
00061     /* Check arguments... */
00062     if (argc < 4)
00063         ERRMSG("Give parameters: <ctl> <tropo.nc> <met0> [ <met1> ... ]");
00064
00065     /* Read control parameters... */
00066     read_ctl(argv[1], argc, argv, &ctl);
00067     lon0 = scan_ctl(argv[1], argc, argv, "TROPO_LON0", -1, "-180", NULL);
00068     lon1 = scan_ctl(argv[1], argc, argv, "TROPO_LON1", -1, "180", NULL);
00069     dlon = scan_ctl(argv[1], argc, argv, "TROPO_DLON", -1, "-999", NULL);
00070     lat0 = scan_ctl(argv[1], argc, argv, "TROPO_LAT0", -1, "-90", NULL);
00071     lat1 = scan_ctl(argv[1], argc, argv, "TROPO_LAT1", -1, "90", NULL);
00072     dlat = scan_ctl(argv[1], argc, argv, "TROPO_DLAT", -1, "-999", NULL);
00073     h2o = (int) scan_ctl(argv[1], argc, argv, "TROPO_H2O", -1, "1", NULL);
00074
00075     /* Loop over files... */
00076     for (i = 3; i < argc; i++) {
00077
00078         /* Read meteorological data... */
00079         ctl.met_tropo = 0;
00080         if (!read_met(&ctl, argv[i], met))
00081             continue;
00082
00083         /* Set horizontal grid... */
00084         if (!init) {
00085             init = 1;
00086
00087             /* Get grid... */
00088             if (dlon <= 0)
00089                 dlon = fabs(met->lon[1] - met->lon[0]);
00090             if (dlat <= 0)
00091                 dlat = fabs(met->lat[1] - met->lat[0]);
00092             if (lon0 < -360 && lon1 > 360) {
00093                 lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00094                 lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00095             }
00096             nx = ny = 0;
00097             for (lon = lon0; lon <= lon1; lon += dlon) {
00098                 lons[nx] = lon;
00099             }
00100         }
00101     }
00102 }
```

```

00099         if ((++nx) > EX)
00100             ERRMSG("Too many longitudes!");
00101     }
00102     if (lat0 < -90 && lat1 > 90) {
00103         lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00104         lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00105     }
00106     for (lat = lat0; lat <= lat1; lat += dlat) {
00107         lats[ny] = lat;
00108         if ((++ny) > EY)
00109             ERRMSG("Too many latitudes!");
00110     }
00111
00112     /* Create netCDF file... */
00113     printf("Write tropopause data file: %s\n", argv[2]);
00114     NC(nc_create(argv[2], NC_CLOBBER, &ncid));
00115
00116     /* Create dimensions... */
00117     NC(nc_def_dim(ncid, "time", (size_t) NC_UNLIMITED, &dims[0]));
00118     NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[1]));
00119     NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[2]));
00120
00121     /* Create variables... */
00122     NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));
00123     NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[1], &latid));
00124     NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[2], &lonid));
00125     NC(nc_def_var(ncid, "clp_z", NC_FLOAT, 3, &dims[0], &clpzid));
00126     NC(nc_def_var(ncid, "clp_p", NC_FLOAT, 3, &dims[0], &clppid));
00127     NC(nc_def_var(ncid, "clp_t", NC_FLOAT, 3, &dims[0], &clptid));
00128     if (h2o)
00129         NC(nc_def_var(ncid, "clp_q", NC_FLOAT, 3, &dims[0], &clpqid));
00130     NC(nc_def_var(ncid, "dyn_z", NC_FLOAT, 3, &dims[0], &dynzid));
00131     NC(nc_def_var(ncid, "dyn_p", NC_FLOAT, 3, &dims[0], &dynpid));
00132     NC(nc_def_var(ncid, "dyn_t", NC_FLOAT, 3, &dims[0], &dyntid));
00133     if (h2o)
00134         NC(nc_def_var(ncid, "dyn_q", NC_FLOAT, 3, &dims[0], &dynqid));
00135     NC(nc_def_var(ncid, "wmo_1st_z", NC_FLOAT, 3, &dims[0], &wmo1zid));
00136     NC(nc_def_var(ncid, "wmo_1st_p", NC_FLOAT, 3, &dims[0], &wmo1pid));
00137     NC(nc_def_var(ncid, "wmo_1st_t", NC_FLOAT, 3, &dims[0], &wmo1tid));
00138     if (h2o)
00139         NC(nc_def_var(ncid, "wmo_1st_q", NC_FLOAT, 3, &dims[0], &wmo1qid));
00140     NC(nc_def_var(ncid, "wmo_2nd_z", NC_FLOAT, 3, &dims[0], &wmo2zid));
00141     NC(nc_def_var(ncid, "wmo_2nd_p", NC_FLOAT, 3, &dims[0], &wmo2pid));
00142     NC(nc_def_var(ncid, "wmo_2nd_t", NC_FLOAT, 3, &dims[0], &wmo2tid));
00143     if (h2o)
00144         NC(nc_def_var(ncid, "wmo_2nd_q", NC_FLOAT, 3, &dims[0], &wmo2qid));
00145
00146     /* Set attributes... */
00147     add_text_attribute(ncid, "time", "units",
00148         "seconds since 2000-01-01 00:00:00 UTC");
00149     add_text_attribute(ncid, "time", "long_name", "time");
00150     add_text_attribute(ncid, "lon", "units", "degrees_east");
00151     add_text_attribute(ncid, "lon", "long_name", "longitude");
00152     add_text_attribute(ncid, "lat", "units", "degrees_north");
00153     add_text_attribute(ncid, "lat", "long_name", "latitude");
00154
00155     add_text_attribute(ncid, "clp_z", "units", "km");
00156     add_text_attribute(ncid, "clp_z", "long_name", "cold point height");
00157     add_text_attribute(ncid, "clp_p", "units", "hPa");
00158     add_text_attribute(ncid, "clp_p", "long_name", "cold point pressure");
00159     add_text_attribute(ncid, "clp_t", "units", "K");
00160     add_text_attribute(ncid, "clp_t", "long_name",
00161         "cold point temperature");
00162     if (h2o) {
00163         add_text_attribute(ncid, "clp_q", "units", "ppv");
00164         add_text_attribute(ncid, "clp_q", "long_name",
00165             "cold point water vapor");
00166     }
00167
00168     add_text_attribute(ncid, "dyn_z", "units", "km");
00169     add_text_attribute(ncid, "dyn_z", "long_name",
00170         "dynamical tropopause height");
00171     add_text_attribute(ncid, "dyn_p", "units", "hPa");
00172     add_text_attribute(ncid, "dyn_p", "long_name",
00173         "dynamical tropopause pressure");
00174     add_text_attribute(ncid, "dyn_t", "units", "K");
00175     add_text_attribute(ncid, "dyn_t", "long_name",
00176         "dynamical tropopause temperature");
00177     if (h2o) {
00178         add_text_attribute(ncid, "dyn_q", "units", "ppv");
00179         add_text_attribute(ncid, "dyn_q", "long_name",
00180             "dynamical tropopause water vapor");
00181     }
00182
00183     add_text_attribute(ncid, "wmo_1st_z", "units", "km");
00184     add_text_attribute(ncid, "wmo_1st_z", "long_name",
00185         "WMO 1st tropopause height");

```

```

00186     add_text_attribute(ncid, "wmo_1st_p", "units", "hPa");
00187     add_text_attribute(ncid, "wmo_1st_p", "long_name",
00188         "WMO 1st tropopause pressure");
00189     add_text_attribute(ncid, "wmo_1st_t", "units", "K");
00190     add_text_attribute(ncid, "wmo_1st_t", "long_name",
00191         "WMO 1st tropopause temperature");
00192     if (h2o) {
00193         add_text_attribute(ncid, "wmo_1st_q", "units", "ppv");
00194         add_text_attribute(ncid, "wmo_1st_q", "long_name",
00195             "WMO 1st tropopause water vapor");
00196     }
00197
00198     add_text_attribute(ncid, "wmo_2nd_z", "units", "km");
00199     add_text_attribute(ncid, "wmo_2nd_z", "long_name",
00200         "WMO 2nd tropopause height");
00201     add_text_attribute(ncid, "wmo_2nd_p", "units", "hPa");
00202     add_text_attribute(ncid, "wmo_2nd_p", "long_name",
00203         "WMO 2nd tropopause pressure");
00204     add_text_attribute(ncid, "wmo_2nd_t", "units", "K");
00205     add_text_attribute(ncid, "wmo_2nd_t", "long_name",
00206         "WMO 2nd tropopause temperature");
00207     if (h2o) {
00208         add_text_attribute(ncid, "wmo_2nd_q", "units", "ppv");
00209         add_text_attribute(ncid, "wmo_2nd_q", "long_name",
00210             "WMO 2nd tropopause water vapor");
00211     }
00212
00213     /* End definition... */
00214     NC(nc_enddef(ncid));
00215
00216     /* Write longitude and latitude... */
00217     NC(nc_put_var_double(ncid, latid, lats));
00218     NC(nc_put_var_double(ncid, lonid, lons));
00219 }
00220
00221 /* Write time... */
00222 start[0] = (size_t) nt;
00223 count[0] = 1;
00224 start[1] = 0;
00225 count[1] = (size_t) ny;
00226 start[2] = 0;
00227 count[2] = (size_t) nx;
00228 NC(nc_put_vara_double(ncid, timid, start, count, &met->time));
00229
00230 /* Get cold point... */
00231 ctl.met_tropo = 2;
00232 read_met_tropo(&ctl, met);
00233 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00234 for (ix = 0; ix < nx; ix++)
00235     for (iy = 0; iy < ny; iy++) {
00236         intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00237             &pt[iy * nx + ix], ci, cw, 1);
00238         intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00239             lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00240         intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00241             lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00242         intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00243             lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00244     }
00245
00246 /* Write data... */
00247 NC(nc_put_vara_double(ncid, clpzid, start, count, zt));
00248 NC(nc_put_vara_double(ncid, clppid, start, count, pt));
00249 NC(nc_put_vara_double(ncid, clptid, start, count, tt));
00250 if (h2o)
00251     NC(nc_put_vara_double(ncid, clpqid, start, count, qt));
00252
00253 /* Get dynamical tropopause... */
00254 ctl.met_tropo = 5;
00255 read_met_tropo(&ctl, met);
00256 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00257 for (ix = 0; ix < nx; ix++)
00258     for (iy = 0; iy < ny; iy++) {
00259         intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00260             &pt[iy * nx + ix], ci, cw, 1);
00261         intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00262             lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00263         intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00264             lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00265         intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00266             lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00267     }
00268
00269 /* Write data... */
00270 NC(nc_put_vara_double(ncid, dynzid, start, count, zt));
00271 NC(nc_put_vara_double(ncid, dynpid, start, count, pt));
00272 NC(nc_put_vara_double(ncid, dyntid, start, count, tt));

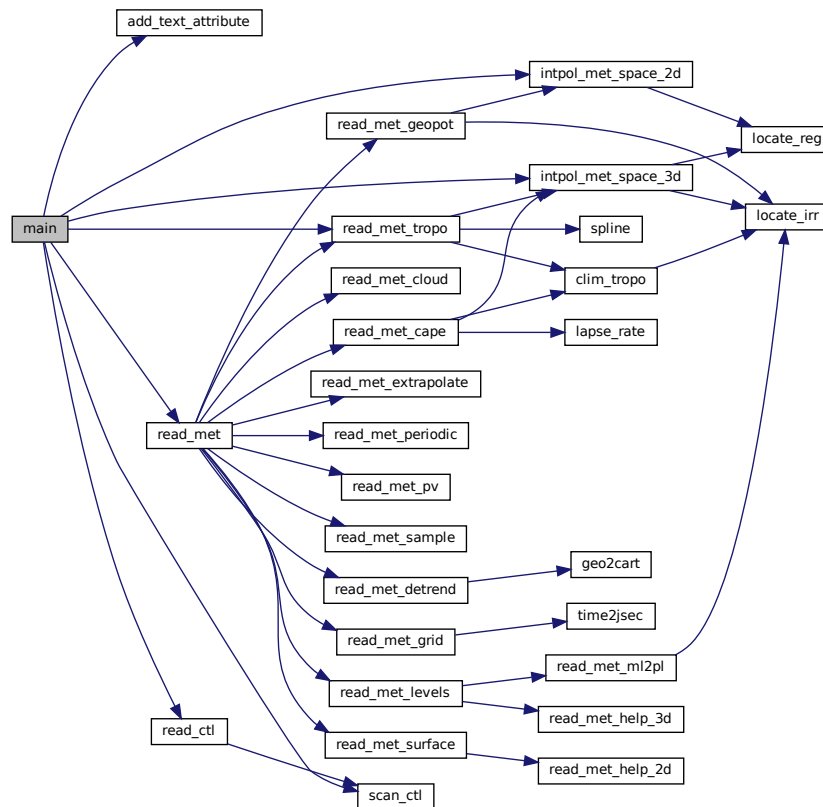
```

```

00273     if (h2o)
00274         NC(nc_put_vara_double(ncid, dynqid, start, count, qt));
00275
00276     /* Get WMO 1st tropopause... */
00277     ctl.met_tropo = 3;
00278     read_met_tropo(&ctl, met);
00279 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00280     for (ix = 0; ix < nx; ix++)
00281         for (iy = 0; iy < ny; iy++) {
00282             intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00283                                 &pt[iy * nx + ix], ci, cw, 1);
00284             intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00285                                 lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00286             intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00287                                 lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00288             intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00289                                 lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00290         }
00291
00292     /* Write data... */
00293     NC(nc_put_vara_double(ncid, wmolzid, start, count, zt));
00294     NC(nc_put_vara_double(ncid, wmolpid, start, count, pt));
00295     NC(nc_put_vara_double(ncid, wmoltid, start, count, tt));
00296     if (h2o)
00297         NC(nc_put_vara_double(ncid, wmolqid, start, count, qt));
00298
00299     /* Get WMO 2nd tropopause... */
00300     ctl.met_tropo = 4;
00301     read_met_tropo(&ctl, met);
00302 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00303     for (ix = 0; ix < nx; ix++)
00304         for (iy = 0; iy < ny; iy++) {
00305             intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00306                                 &pt[iy * nx + ix], ci, cw, 1);
00307             intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00308                                 lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00309             intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00310                                 lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00311             intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00312                                 lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00313         }
00314
00315     /* Write data... */
00316     NC(nc_put_vara_double(ncid, wmo2zid, start, count, zt));
00317     NC(nc_put_vara_double(ncid, wmo2pid, start, count, pt));
00318     NC(nc_put_vara_double(ncid, wmo2tid, start, count, tt));
00319     if (h2o)
00320         NC(nc_put_vara_double(ncid, wmo2qid, start, count, qt));
00321
00322     /* Increment time step counter... */
00323     nt++;
00324 }
00325
00326 /* Close file... */
00327 NC(nc_close(ncid));
00328
00329 /* Free... */
00330 free(met);
00331
00332 return EXIT_SUCCESS;
00333 }

```

Here is the call graph for this function:



## 5.40 tropo.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
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2021 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,
00034     char *attrname,
00035     char *text);
00036
00037 /* -----
00038  Main...
00039  ----- */
  
```

```

00040
00041 int main(
00042     int argc,
00043     char *argv[]) {
00044
00045     ctl_t ctl;
00046
00047     met_t *met;
00048
00049     static double pt[EX * EY], qt[EX * EY], zt[EX * EY], tt[EX * EY], lon, lon0,
00050         lon1, lons[EX], dlon, lat, lat0, lat1, lats[EY], dlat, cw[3];
00051
00052     static int init, i, ix, iy, nx, ny, nt, ncid, dims[3], timid, lonid, latid,
00053         clppid, clpqid, clptid, clpzid, dynpid, dynqid, dyntid, dynzid, wmolpid,
00054         wmolqid, wmoltid, wmolzid, wmo2pid, wmo2qid, wmo2tid, wmo2zid, h2o, ci[3];
00055
00056     static size_t count[10], start[10];
00057
00058     /* Allocate... */
00059     ALLOC(met, met_t, 1);
00060
00061     /* Check arguments... */
00062     if (argc < 4)
00063         ERRMSG("Give parameters: <ctl> <tropo.nc> <met0> [ <met1> ... ]");
00064
00065     /* Read control parameters... */
00066     read_ctl(argv[1], argc, argv, &ctl);
00067     lon0 = scan_ctl(argv[1], argc, argv, "TROPO_LON0", -1, "-180", NULL);
00068     lon1 = scan_ctl(argv[1], argc, argv, "TROPO_LON1", -1, "180", NULL);
00069     dlon = scan_ctl(argv[1], argc, argv, "TROPO_DLON", -1, "-999", NULL);
00070     lat0 = scan_ctl(argv[1], argc, argv, "TROPO_LAT0", -1, "-90", NULL);
00071     lat1 = scan_ctl(argv[1], argc, argv, "TROPO_LAT1", -1, "90", NULL);
00072     dlat = scan_ctl(argv[1], argc, argv, "TROPO_DLAT", -1, "-999", NULL);
00073     h2o = (int) scan_ctl(argv[1], argc, argv, "TROPO_H2O", -1, "1", NULL);
00074
00075     /* Loop over files... */
00076     for (i = 3; i < argc; i++) {
00077
00078         /* Read meteorological data... */
00079         ctl.met_tropo = 0;
00080         if (!read_met(&ctl, argv[i], met))
00081             continue;
00082
00083         /* Set horizontal grid... */
00084         if (!init) {
00085             init = 1;
00086
00087             /* Get grid... */
00088             if (dlon <= 0)
00089                 dlon = fabs(met->lon[1] - met->lon[0]);
00090             if (dlat <= 0)
00091                 dlat = fabs(met->lat[1] - met->lat[0]);
00092             if (lon0 < -360 && lon1 > 360) {
00093                 lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00094                 lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00095             }
00096             nx = ny = 0;
00097             for (lon = lon0; lon <= lon1; lon += dlon) {
00098                 lons[nx] = lon;
00099                 if ((++nx) > EX)
00100                     ERRMSG("Too many longitudes!");
00101             }
00102             if (lat0 < -90 && lat1 > 90) {
00103                 lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00104                 lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00105             }
00106             for (lat = lat0; lat <= lat1; lat += dlat) {
00107                 lats[ny] = lat;
00108                 if ((++ny) > EY)
00109                     ERRMSG("Too many latitudes!");
00110             }
00111
00112             /* Create netCDF file... */
00113             printf("Write tropopause data file: %s\n", argv[2]);
00114             NC(nc_create(argv[2], NC_CLOBBER, &ncid));
00115
00116             /* Create dimensions... */
00117             NC(nc_def_dim(ncid, "time", (size_t) NC_UNLIMITED, &dims[0]));
00118             NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[1]));
00119             NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[2]));
00120
00121             /* Create variables... */
00122             NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));
00123             NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[1], &latid));
00124             NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[2], &lonid));
00125             NC(nc_def_var(ncid, "clp_z", NC_FLOAT, 3, &dims[0], &clpzid));
00126             NC(nc_def_var(ncid, "clp_p", NC_FLOAT, 3, &dims[0], &clppid));

```

```

00127     NC(nc_def_var(ncid, "clp_t", NC_FLOAT, 3, &dims[0], &clptid));
00128     if (h2o)
00129         NC(nc_def_var(ncid, "clp_q", NC_FLOAT, 3, &dims[0], &clpqid));
00130     NC(nc_def_var(ncid, "dyn_z", NC_FLOAT, 3, &dims[0], &dynzid));
00131     NC(nc_def_var(ncid, "dyn_p", NC_FLOAT, 3, &dims[0], &dynpid));
00132     NC(nc_def_var(ncid, "dyn_t", NC_FLOAT, 3, &dims[0], &dyntid));
00133     if (h2o)
00134         NC(nc_def_var(ncid, "dyn_q", NC_FLOAT, 3, &dims[0], &dynqid));
00135     NC(nc_def_var(ncid, "wmo_1st_z", NC_FLOAT, 3, &dims[0], &wmo1zid));
00136     NC(nc_def_var(ncid, "wmo_1st_p", NC_FLOAT, 3, &dims[0], &wmo1pid));
00137     NC(nc_def_var(ncid, "wmo_1st_t", NC_FLOAT, 3, &dims[0], &wmo1tid));
00138     if (h2o)
00139         NC(nc_def_var(ncid, "wmo_1st_q", NC_FLOAT, 3, &dims[0], &wmo1qid));
00140     NC(nc_def_var(ncid, "wmo_2nd_z", NC_FLOAT, 3, &dims[0], &wmo2zid));
00141     NC(nc_def_var(ncid, "wmo_2nd_p", NC_FLOAT, 3, &dims[0], &wmo2pid));
00142     NC(nc_def_var(ncid, "wmo_2nd_t", NC_FLOAT, 3, &dims[0], &wmo2tid));
00143     if (h2o)
00144         NC(nc_def_var(ncid, "wmo_2nd_q", NC_FLOAT, 3, &dims[0], &wmo2qid));
00145
00146     /* Set attributes... */
00147     add_text_attribute(ncid, "time", "units",
00148         "seconds since 2000-01-01 00:00:00 UTC");
00149     add_text_attribute(ncid, "time", "long_name", "time");
00150     add_text_attribute(ncid, "lon", "units", "degrees_east");
00151     add_text_attribute(ncid, "lon", "long_name", "longitude");
00152     add_text_attribute(ncid, "lat", "units", "degrees_north");
00153     add_text_attribute(ncid, "lat", "long_name", "latitude");
00154
00155     add_text_attribute(ncid, "clp_z", "units", "km");
00156     add_text_attribute(ncid, "clp_z", "long_name", "cold point height");
00157     add_text_attribute(ncid, "clp_p", "units", "hPa");
00158     add_text_attribute(ncid, "clp_p", "long_name", "cold point pressure");
00159     add_text_attribute(ncid, "clp_t", "units", "K");
00160     add_text_attribute(ncid, "clp_t", "long_name",
00161         "cold point temperature");
00162     if (h2o) {
00163         add_text_attribute(ncid, "clp_q", "units", "ppv");
00164         add_text_attribute(ncid, "clp_q", "long_name",
00165             "cold point water vapor");
00166     }
00167
00168     add_text_attribute(ncid, "dyn_z", "units", "km");
00169     add_text_attribute(ncid, "dyn_z", "long_name",
00170         "dynamical tropopause height");
00171     add_text_attribute(ncid, "dyn_p", "units", "hPa");
00172     add_text_attribute(ncid, "dyn_p", "long_name",
00173         "dynamical tropopause pressure");
00174     add_text_attribute(ncid, "dyn_t", "units", "K");
00175     add_text_attribute(ncid, "dyn_t", "long_name",
00176         "dynamical tropopause temperature");
00177     if (h2o) {
00178         add_text_attribute(ncid, "dyn_q", "units", "ppv");
00179         add_text_attribute(ncid, "dyn_q", "long_name",
00180             "dynamical tropopause water vapor");
00181     }
00182
00183     add_text_attribute(ncid, "wmo_1st_z", "units", "km");
00184     add_text_attribute(ncid, "wmo_1st_z", "long_name",
00185         "WMO 1st tropopause height");
00186     add_text_attribute(ncid, "wmo_1st_p", "units", "hPa");
00187     add_text_attribute(ncid, "wmo_1st_p", "long_name",
00188         "WMO 1st tropopause pressure");
00189     add_text_attribute(ncid, "wmo_1st_t", "units", "K");
00190     add_text_attribute(ncid, "wmo_1st_t", "long_name",
00191         "WMO 1st tropopause temperature");
00192     if (h2o) {
00193         add_text_attribute(ncid, "wmo_1st_q", "units", "ppv");
00194         add_text_attribute(ncid, "wmo_1st_q", "long_name",
00195             "WMO 1st tropopause water vapor");
00196     }
00197
00198     add_text_attribute(ncid, "wmo_2nd_z", "units", "km");
00199     add_text_attribute(ncid, "wmo_2nd_z", "long_name",
00200         "WMO 2nd tropopause height");
00201     add_text_attribute(ncid, "wmo_2nd_p", "units", "hPa");
00202     add_text_attribute(ncid, "wmo_2nd_p", "long_name",
00203         "WMO 2nd tropopause pressure");
00204     add_text_attribute(ncid, "wmo_2nd_t", "units", "K");
00205     add_text_attribute(ncid, "wmo_2nd_t", "long_name",
00206         "WMO 2nd tropopause temperature");
00207     if (h2o) {
00208         add_text_attribute(ncid, "wmo_2nd_q", "units", "ppv");
00209         add_text_attribute(ncid, "wmo_2nd_q", "long_name",
00210             "WMO 2nd tropopause water vapor");
00211     }
00212
00213     /* End definition... */

```



```

00214     NC(nc_enddef(ncid));
00215
00216     /* Write longitude and latitude... */
00217     NC(nc_put_var_double(ncid, latid, lats));
00218     NC(nc_put_var_double(ncid, lonid, lons));
00219 }
00220
00221 /* Write time... */
00222 start[0] = (size_t) nt;
00223 count[0] = 1;
00224 start[1] = 0;
00225 count[1] = (size_t) ny;
00226 start[2] = 0;
00227 count[2] = (size_t) nx;
00228 NC(nc_put_vara_double(ncid, timid, start, count, &met->time));
00229
00230 /* Get cold point... */
00231 ctl.met_tropo = 2;
00232 read_met_tropo(&ctl, met);
00233 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00234 for (ix = 0; ix < nx; ix++)
00235     for (iy = 0; iy < ny; iy++) {
00236         intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00237                             &pt[iy * nx + ix], ci, cw, 1);
00238         intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00239                             lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00240         intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00241                             lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00242         intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00243                             lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00244     }
00245
00246 /* Write data... */
00247 NC(nc_put_vara_double(ncid, clpzid, start, count, zt));
00248 NC(nc_put_vara_double(ncid, clppid, start, count, pt));
00249 NC(nc_put_vara_double(ncid, clptid, start, count, tt));
00250 if (h2o)
00251     NC(nc_put_vara_double(ncid, clpqid, start, count, qt));
00252
00253 /* Get dynamical tropopause... */
00254 ctl.met_tropo = 5;
00255 read_met_tropo(&ctl, met);
00256 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00257 for (ix = 0; ix < nx; ix++)
00258     for (iy = 0; iy < ny; iy++) {
00259         intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00260                             &pt[iy * nx + ix], ci, cw, 1);
00261         intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00262                             lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00263         intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00264                             lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00265         intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00266                             lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00267     }
00268
00269 /* Write data... */
00270 NC(nc_put_vara_double(ncid, dynzid, start, count, zt));
00271 NC(nc_put_vara_double(ncid, dynpid, start, count, pt));
00272 NC(nc_put_vara_double(ncid, dyntid, start, count, tt));
00273 if (h2o)
00274     NC(nc_put_vara_double(ncid, dynqid, start, count, qt));
00275
00276 /* Get WMO 1st tropopause... */
00277 ctl.met_tropo = 3;
00278 read_met_tropo(&ctl, met);
00279 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00280 for (ix = 0; ix < nx; ix++)
00281     for (iy = 0; iy < ny; iy++) {
00282         intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00283                             &pt[iy * nx + ix], ci, cw, 1);
00284         intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00285                             lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00286         intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00287                             lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00288         intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00289                             lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00290     }
00291
00292 /* Write data... */
00293 NC(nc_put_vara_double(ncid, wmolzid, start, count, zt));
00294 NC(nc_put_vara_double(ncid, wmolpid, start, count, pt));
00295 NC(nc_put_vara_double(ncid, wmoltid, start, count, tt));
00296 if (h2o)
00297     NC(nc_put_vara_double(ncid, wmolqid, start, count, qt));
00298
00299 /* Get WMO 2nd tropopause... */
00300 ctl.met_tropo = 4;

```

```

00301     read_met_tropo(&ctl, met);
00302 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00303     for (ix = 0; ix < nx; ix++)
00304         for (iy = 0; iy < ny; iy++) {
00305             intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00306                                 &pt[iy * nx + ix], ci, cw, 1);
00307             intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00308                                 lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00309             intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00310                                 lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00311             intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00312                                 lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00313         }
00314
00315     /* Write data... */
00316     NC(nc_put_vara_double(ncid, wmo2zid, start, count, zt));
00317     NC(nc_put_vara_double(ncid, wmo2pid, start, count, pt));
00318     NC(nc_put_vara_double(ncid, wmo2tid, start, count, tt));
00319     if (h2o)
00320         NC(nc_put_vara_double(ncid, wmo2qid, start, count, qt));
00321
00322     /* Increment time step counter... */
00323     nt++;
00324 }
00325
00326 /* Close file... */
00327 NC(nc_close(ncid));
00328
00329 /* Free... */
00330 free(met);
00331
00332 return EXIT_SUCCESS;
00333 }
00334
00335 /*****
00336
00337 void add_text_attribute(
00338     int ncid,
00339     char *varname,
00340     char *attrname,
00341     char *text) {
00342
00343     int varid;
00344
00345     NC(nc_inq_varid(ncid, varname, &varid));
00346     NC(nc_put_att_text(ncid, varid, attrname, strlen(text), text));
00347 }

```

## 5.41 tropo\_sample.c File Reference

### Functions

- void [intpol\\_tropo\\_3d](#) (double time0, float array0[EX][EY], double time1, float array1[EX][EY], double lons[EX], double lats[EY], size\_t nlon, size\_t nlat, double time, double lon, double lat, int method, double \*var, double \*sigma)  
*3-D linear interpolation of tropopause data.*
- int [main](#) (int argc, char \*argv[])

#### 5.41.1 Detailed Description

Sample tropopause data set.

Definition in file [tropo\\_sample.c](#).

#### 5.41.2 Function Documentation

```

5.41.2.1 intpol_tropo_3d() void intpol_tropo_3d (
    double time0,
    float array0[EX][EY],
    double time1,
    float array1[EX][EY],
    double lons[EX],
    double lats[EY],
    size_t nlon,
    size_t nlat,
    double time,
    double lon,
    double lat,
    int method,
    double * var,
    double * sigma )

```

3-D linear interpolation of tropopause data.

Definition at line 266 of file `tropo_sample.c`.

```

00280         {
00281
00282     double aux0, aux1, aux00, aux01, aux10, aux11, mean = 0;
00283
00284     int n = 0;
00285
00286     /* Adjust longitude... */
00287     if (lon < lons[0])
00288         lon += 360;
00289     else if (lon > lons[nlon - 1])
00290         lon -= 360;
00291
00292     /* Get indices... */
00293     int ix = locate_reg(lons, (int) nlon, lon);
00294     int iy = locate_reg(lats, (int) nlat, lat);
00295
00296     /* Calculate standard deviation... */
00297     *sigma = 0;
00298     for (int dx = 0; dx < 2; dx++)
00299         for (int dy = 0; dy < 2; dy++) {
00300             if (isfinite(array0[ix + dx][iy + dy])) {
00301                 mean += array0[ix + dx][iy + dy];
00302                 *sigma += SQR(array0[ix + dx][iy + dy]);
00303                 n++;
00304             }
00305             if (isfinite(array1[ix + dx][iy + dy])) {
00306                 mean += array1[ix + dx][iy + dy];
00307                 *sigma += SQR(array1[ix + dx][iy + dy]);
00308                 n++;
00309             }
00310         }
00311     if (n > 0)
00312         *sigma = sqrt(GSL_MAX(*sigma / n - SQR(mean / n), 0.0));
00313
00314     /* Linear interpolation... */
00315     if (method == 1 && isfinite(array0[ix][iy])
00316         && isfinite(array0[ix][iy + 1])
00317         && isfinite(array0[ix + 1][iy])
00318         && isfinite(array0[ix + 1][iy + 1])
00319         && isfinite(array1[ix][iy])
00320         && isfinite(array1[ix][iy + 1])
00321         && isfinite(array1[ix + 1][iy])
00322         && isfinite(array1[ix + 1][iy + 1])) {
00323
00324         aux00 = LIN(lons[ix], array0[ix][iy],
00325                    lons[ix + 1], array0[ix + 1][iy], lon);
00326         aux01 = LIN(lons[ix], array0[ix][iy + 1],
00327                    lons[ix + 1], array0[ix + 1][iy + 1], lon);
00328         aux0 = LIN(lats[iy], aux00, lats[iy + 1], aux01, lat);
00329
00330         aux10 = LIN(lons[ix], array1[ix][iy],
00331                    lons[ix + 1], array1[ix + 1][iy], lon);
00332         aux11 = LIN(lons[ix], array1[ix][iy + 1],
00333                    lons[ix + 1], array1[ix + 1][iy + 1], lon);
00334         aux1 = LIN(lats[iy], aux10, lats[iy + 1], aux11, lat);
00335
00336         *var = LIN(time0, aux0, time1, aux1, time);
00337     }
00338

```

```

00339  /* Nearest neighbor interpolation... */
00340  else {
00341      aux00 = NN(lons[ix], array0[ix][iy],
00342                lons[ix + 1], array0[ix + 1][iy], lon);
00343      aux01 = NN(lons[ix], array0[ix][iy + 1],
00344                lons[ix + 1], array0[ix + 1][iy + 1], lon);
00345      aux0 = NN(lats[iy], aux00, lats[iy + 1], aux01, lat);
00346
00347      aux10 = NN(lons[ix], array1[ix][iy],
00348                lons[ix + 1], array1[ix + 1][iy], lon);
00349      aux11 = NN(lons[ix], array1[ix][iy + 1],
00350                lons[ix + 1], array1[ix + 1][iy + 1], lon);
00351      aux1 = NN(lats[iy], aux10, lats[iy + 1], aux11, lat);
00352
00353      *var = NN(time0, aux0, time1, aux1, time);
00354  }
00355 }

```

Here is the call graph for this function:



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

Definition at line 59 of file tropo\_sample.c.

```

00061      {
00062
00063      ctl_t ctl;
00064
00065      atm_t *atm;
00066
00067      static FILE *out;
00068
00069      static char varname[LEN];
00070
00071      static double times[NT], lons[EX], lats[EY], time0, time1, z0, z0sig,
00072      p0, p0sig, t0, t0sig, q0, q0sig;
00073
00074      static float help[EX * EY], tropo_z0[EX][EY], tropo_z1[EX][EY],
00075      tropo_p0[EX][EY], tropo_p1[EX][EY], tropo_t0[EX][EY],
00076      tropo_t1[EX][EY], tropo_q0[EX][EY], tropo_q1[EX][EY];
00077
00078      static int ip, iq, it, it_old = -999, method, dimid[10], ncid,
00079      varid, varid_z, varid_p, varid_t, varid_q, h2o;
00080
00081      static size_t count[10], start[10], ntime, nlon, nlat, ilon, ilat;
00082
00083      /* Allocate... */
00084      ALLOC(atm, atm_t, 1);
00085
00086      /* Check arguments... */
00087      if (argc < 5)
00088          ERRMSG("Give parameters: <ctl> <sample.tab> <tropo.nc> <var> <atm_in>");
00089
00090      /* Read control parameters... */
00091      read_ctl(argv[1], argc, argv, &ctl);
00092      method =
00093          (int) scan_ctl(argv[1], argc, argv, "TROPO_SAMPLE_METHOD", -1, "1", NULL);
00094
00095      /* Read atmospheric data... */
00096      if (!read_atm(argv[5], &ctl, atm))

```

```

00097     ERRMSG("Cannot open file!");
00098
00099     /* Open tropopause file... */
00100     printf("Read tropopause data: %s\n", argv[3]);
00101     if (nc_open(argv[3], NC_NOWRITE, &ncid) != NC_NOERR)
00102         ERRMSG("Cannot open file!");
00103
00104     /* Get dimensions... */
00105     NC(nc_inq_dimid(ncid, "time", &dimid[0]));
00106     NC(nc_inq_dimlen(ncid, dimid[0], &ntime));
00107     if (ntime > NT)
00108         ERRMSG("Too many times!");
00109     NC(nc_inq_dimid(ncid, "lat", &dimid[1]));
00110     NC(nc_inq_dimlen(ncid, dimid[1], &nlat));
00111     if (nlat > EY)
00112         ERRMSG("Too many latitudes!");
00113     NC(nc_inq_dimid(ncid, "lon", &dimid[2]));
00114     NC(nc_inq_dimlen(ncid, dimid[2], &nlon));
00115     if (nlon > EX)
00116         ERRMSG("Too many longitudes!");
00117
00118     /* Read coordinates... */
00119     NC(nc_inq_varid(ncid, "time", &varid));
00120     NC(nc_get_var_double(ncid, varid, times));
00121     NC(nc_inq_varid(ncid, "lat", &varid));
00122     NC(nc_get_var_double(ncid, varid, lats));
00123     NC(nc_inq_varid(ncid, "lon", &varid));
00124     NC(nc_get_var_double(ncid, varid, lons));
00125
00126     /* Get variable indices... */
00127     sprintf(varname, "%s_z", argv[4]);
00128     NC(nc_inq_varid(ncid, varname, &varid_z));
00129     sprintf(varname, "%s_p", argv[4]);
00130     NC(nc_inq_varid(ncid, varname, &varid_p));
00131     sprintf(varname, "%s_t", argv[4]);
00132     NC(nc_inq_varid(ncid, varname, &varid_t));
00133     sprintf(varname, "%s_q", argv[4]);
00134     h2o = (nc_inq_varid(ncid, varname, &varid_q) == NC_NOERR);
00135
00136     /* Set dimensions... */
00137     count[0] = 1;
00138     count[1] = nlat;
00139     count[2] = nlon;
00140
00141     /* Create file... */
00142     printf("Write tropopause sample data: %s\n", argv[2]);
00143     if (!(out = fopen(argv[2], "w")))
00144         ERRMSG("Cannot create file!");
00145
00146     /* Write header... */
00147     fprintf(out,
00148         "# $1 = time [s]\n"
00149         "# $2 = altitude [km]\n"
00150         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00151     for (iq = 0; iq < ctl.nq; iq++)
00152         fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00153             ctl.qnt_unit[iq]);
00154     fprintf(out, "# $%d = tropopause height [km]\n", 5 + ctl.nq);
00155     fprintf(out, "# $%d = tropopause pressure [hPa]\n", 6 + ctl.nq);
00156     fprintf(out, "# $%d = tropopause temperature [K]\n", 7 + ctl.nq);
00157     fprintf(out, "# $%d = tropopause water vapor [ppv]\n", 8 + ctl.nq);
00158     fprintf(out, "# $%d = tropopause height (sigma) [km]\n", 9 + ctl.nq);
00159     fprintf(out, "# $%d = tropopause pressure (sigma) [hPa]\n", 10 + ctl.nq);
00160     fprintf(out, "# $%d = tropopause temperature (sigma) [K]\n", 11 + ctl.nq);
00161     fprintf(out, "# $%d = tropopause water vapor (sigma) [ppv]\n",
00162         12 + ctl.nq);
00163
00164     /* Loop over particles... */
00165     for (ip = 0; ip < atm->np; ip++) {
00166
00167         /* Check temporal ordering... */
00168         if (ip > 0 && atm->time[ip] < atm->time[ip - 1])
00169             ERRMSG("Time must be ascending!");
00170
00171         /* Check range... */
00172         if (atm->time[ip] < times[0] || atm->time[ip] > times[ntime - 1])
00173             continue;
00174
00175         /* Read data... */
00176         it = locate_irr(times, (int) ntime, atm->time[ip]);
00177         if (it != it_old) {
00178
00179             time0 = times[it];
00180             start[0] = (size_t) it;
00181             NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00182             for (ilon = 0; ilon < nlon; ilon++)
00183                 for (ilat = 0; ilat < nlat; ilat++)

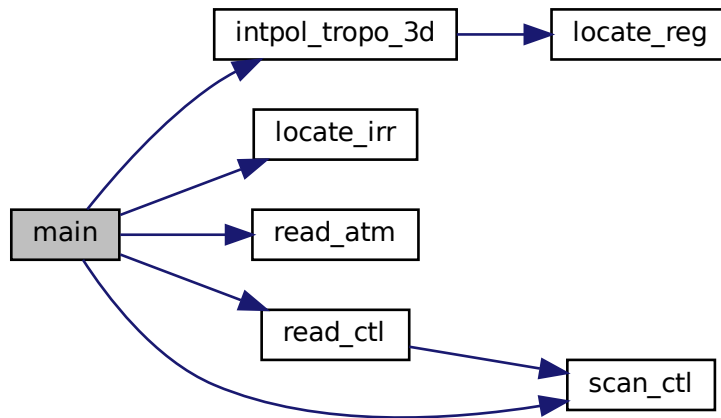
```

```

00184     tropo_z0[ilon][ilat] = help[ilat * nlon + ilon];
00185     NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00186     for (ilon = 0; ilon < nlon; ilon++)
00187         for (ilat = 0; ilat < nlat; ilat++)
00188             tropo_p0[ilon][ilat] = help[ilat * nlon + ilon];
00189     NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00190     for (ilon = 0; ilon < nlon; ilon++)
00191         for (ilat = 0; ilat < nlat; ilat++)
00192             tropo_t0[ilon][ilat] = help[ilat * nlon + ilon];
00193     if (h2o) {
00194         NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00195         for (ilon = 0; ilon < nlon; ilon++)
00196             for (ilat = 0; ilat < nlat; ilat++)
00197                 tropo_q0[ilon][ilat] = help[ilat * nlon + ilon];
00198     } else
00199         for (ilon = 0; ilon < nlon; ilon++)
00200             for (ilat = 0; ilat < nlat; ilat++)
00201                 tropo_q0[ilon][ilat] = GSL_NAN;
00202
00203     time1 = times[it + 1];
00204     start[0] = (size_t) it + 1;
00205     NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00206     for (ilon = 0; ilon < nlon; ilon++)
00207         for (ilat = 0; ilat < nlat; ilat++)
00208             tropo_z1[ilon][ilat] = help[ilat * nlon + ilon];
00209     NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00210     for (ilon = 0; ilon < nlon; ilon++)
00211         for (ilat = 0; ilat < nlat; ilat++)
00212             tropo_p1[ilon][ilat] = help[ilat * nlon + ilon];
00213     NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00214     for (ilon = 0; ilon < nlon; ilon++)
00215         for (ilat = 0; ilat < nlat; ilat++)
00216             tropo_t1[ilon][ilat] = help[ilat * nlon + ilon];
00217     if (h2o) {
00218         NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00219         for (ilon = 0; ilon < nlon; ilon++)
00220             for (ilat = 0; ilat < nlat; ilat++)
00221                 tropo_q1[ilon][ilat] = help[ilat * nlon + ilon];
00222     } else
00223         for (ilon = 0; ilon < nlon; ilon++)
00224             for (ilat = 0; ilat < nlat; ilat++)
00225                 tropo_q1[ilon][ilat] = GSL_NAN;;
00226 }
00227 it_old = it;
00228
00229 /* Interpolate... */
00230 intpol_tropo_3d(time0, tropo_z0, time1, tropo_z1,
00231                lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00232                atm->lat[ip], method, &z0, &z0sig);
00233 intpol_tropo_3d(time0, tropo_p0, time1, tropo_p1,
00234                lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00235                atm->lat[ip], method, &p0, &p0sig);
00236 intpol_tropo_3d(time0, tropo_t0, time1, tropo_t1,
00237                lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00238                atm->lat[ip], method, &t0, &t0sig);
00239 intpol_tropo_3d(time0, tropo_q0, time1, tropo_q1,
00240                lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00241                atm->lat[ip], method, &q0, &q0sig);
00242
00243 /* Write output... */
00244 fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
00245        atm->lon[ip], atm->lat[ip]);
00246 for (iq = 0; iq < ctl.nq; iq++) {
00247     fprintf(out, " ");
00248     fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00249 }
00250 fprintf(out, " %g %g %g %g %g %g %g %g\n",
00251        z0, p0, t0, q0, z0sig, p0sig, t0sig, q0sig);
00252 }
00253
00254 /* Close files... */
00255 fclose(out);
00256 NC(nc_close(ncid));
00257
00258 /* Free... */
00259 free(atm);
00260
00261 return EXIT_SUCCESS;
00262 }

```

Here is the call graph for this function:



## 5.42 tropo\_sample.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
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028  Dimensions...
00029  ----- */
00030
00032 #define NT 744
00033
00034 /* -----
00035  Functions...
00036  ----- */
00037
00039 void intpol_tropo_3d(
00040     double time0,
00041     float array0[EX][EY],
00042     double time1,
00043     float array1[EX][EY],
00044     double lons[EX],
00045     double lats[EY],
00046     size_t nlon,
00047     size_t nlat,
00048     double time,
00049     double lon,
00050     double lat,
00051     int method,
00052     double *var,
00053     double *sigma);
00054
00055 /* -----
  
```

```

00056     Main...
00057     ----- */
00058
00059 int main(
00060     int argc,
00061     char *argv[]) {
00062     ctl_t ctl;
00063
00064     atm_t *atm;
00065
00066     static FILE *out;
00067
00068     static char varname[LEN];
00069
00070     static double times[NT], lons[EX], lats[EY], time0, time1, z0, z0sig,
00071         p0, p0sig, t0, t0sig, q0, q0sig;
00072
00073     static float help[EX * EY], tropo_z0[EX][EY], tropo_z1[EX][EY],
00074         tropo_p0[EX][EY], tropo_p1[EX][EY], tropo_t0[EX][EY],
00075         tropo_t1[EX][EY], tropo_q0[EX][EY], tropo_q1[EX][EY];
00076
00077     static int ip, iq, it, it_old = -999, method, dimid[10], ncid,
00078         varid, varid_z, varid_p, varid_t, varid_q, h2o;
00079
00080     static size_t count[10], start[10], ntime, nlon, nlat, ilon, ilat;
00081
00082     /* Allocate... */
00083     ALLOC(atm, atm_t, 1);
00084
00085     /* Check arguments... */
00086     if (argc < 5)
00087         ERRMSG("Give parameters: <ctl> <sample.tab> <tropo.nc> <var> <atm_in>");
00088
00089     /* Read control parameters... */
00090     read_ctl(argv[1], argc, argv, &ctl);
00091     method =
00092         (int) scan_ctl(argv[1], argc, argv, "TROPO_SAMPLE_METHOD", -1, "1", NULL);
00093
00094     /* Read atmospheric data... */
00095     if (!read_atm(argv[5], &ctl, atm))
00096         ERRMSG("Cannot open file!");
00097
00098     /* Open tropopause file... */
00099     printf("Read tropopause data: %s\n", argv[3]);
00100     if (nc_open(argv[3], NC_NOWRITE, &ncid) != NC_NOERR)
00101         ERRMSG("Cannot open file!");
00102
00103     /* Get dimensions... */
00104     NC(nc_inq_dimid(ncid, "time", &dimid[0]));
00105     NC(nc_inq_dimlen(ncid, dimid[0], &ntime));
00106     if (ntime > NT)
00107         ERRMSG("Too many times!");
00108     NC(nc_inq_dimid(ncid, "lat", &dimid[1]));
00109     NC(nc_inq_dimlen(ncid, dimid[1], &nlat));
00110     if (nlat > EY)
00111         ERRMSG("Too many latitudes!");
00112     NC(nc_inq_dimid(ncid, "lon", &dimid[2]));
00113     NC(nc_inq_dimlen(ncid, dimid[2], &nlon));
00114     if (nlon > EX)
00115         ERRMSG("Too many longitudes!");
00116
00117     /* Read coordinates... */
00118     NC(nc_inq_varid(ncid, "time", &varid));
00119     NC(nc_get_var_double(ncid, varid, times));
00120     NC(nc_inq_varid(ncid, "lat", &varid));
00121     NC(nc_get_var_double(ncid, varid, lats));
00122     NC(nc_inq_varid(ncid, "lon", &varid));
00123     NC(nc_get_var_double(ncid, varid, lons));
00124
00125     /* Get variable indices... */
00126     sprintf(varname, "%s_z", argv[4]);
00127     NC(nc_inq_varid(ncid, varname, &varid_z));
00128     sprintf(varname, "%s_p", argv[4]);
00129     NC(nc_inq_varid(ncid, varname, &varid_p));
00130     sprintf(varname, "%s_t", argv[4]);
00131     NC(nc_inq_varid(ncid, varname, &varid_t));
00132     sprintf(varname, "%s_q", argv[4]);
00133     h2o = (nc_inq_varid(ncid, varname, &varid_q) == NC_NOERR);
00134
00135     /* Set dimensions... */
00136     count[0] = 1;
00137     count[1] = nlat;
00138     count[2] = nlon;
00139
00140     /* Create file... */
00141     printf("Write tropopause sample data: %s\n", argv[2]);

```



```

00143 if (!out = fopen(argv[2], "w"))
00144     ERRMSG("Cannot create file!");
00145
00146 /* Write header... */
00147 fprintf(out,
00148     "# $1 = time [s]\n"
00149     "# $2 = altitude [km]\n"
00150     "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00151 for (iq = 0; iq < ctl.nq; iq++)
00152     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00153         ctl.qnt_unit[iq]);
00154 fprintf(out, "# $%d = tropopause height [km]\n", 5 + ctl.nq);
00155 fprintf(out, "# $%d = tropopause pressure [hPa]\n", 6 + ctl.nq);
00156 fprintf(out, "# $%d = tropopause temperature [K]\n", 7 + ctl.nq);
00157 fprintf(out, "# $%d = tropopause water vapor [ppv]\n", 8 + ctl.nq);
00158 fprintf(out, "# $%d = tropopause height (sigma) [km]\n", 9 + ctl.nq);
00159 fprintf(out, "# $%d = tropopause pressure (sigma) [hPa]\n", 10 + ctl.nq);
00160 fprintf(out, "# $%d = tropopause temperature (sigma) [K]\n", 11 + ctl.nq);
00161 fprintf(out, "# $%d = tropopause water vapor (sigma) [ppv]\n",
00162     12 + ctl.nq);
00163
00164 /* Loop over particles... */
00165 for (ip = 0; ip < atm->np; ip++) {
00166
00167     /* Check temporal ordering... */
00168     if (ip > 0 && atm->time[ip] < atm->time[ip - 1])
00169         ERRMSG("Time must be ascending!");
00170
00171     /* Check range... */
00172     if (atm->time[ip] < times[0] || atm->time[ip] > times[ntime - 1])
00173         continue;
00174
00175     /* Read data... */
00176     it = locate_irr(times, (int) ntime, atm->time[ip]);
00177     if (it != it_old) {
00178
00179         time0 = times[it];
00180         start[0] = (size_t) it;
00181         NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00182         for (ilon = 0; ilon < nlon; ilon++)
00183             for (ilat = 0; ilat < nlat; ilat++)
00184                 tropo_z0[ilon][ilat] = help[ilat * nlon + ilon];
00185         NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00186         for (ilon = 0; ilon < nlon; ilon++)
00187             for (ilat = 0; ilat < nlat; ilat++)
00188                 tropo_p0[ilon][ilat] = help[ilat * nlon + ilon];
00189         NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00190         for (ilon = 0; ilon < nlon; ilon++)
00191             for (ilat = 0; ilat < nlat; ilat++)
00192                 tropo_t0[ilon][ilat] = help[ilat * nlon + ilon];
00193         if (h2o) {
00194             NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00195             for (ilon = 0; ilon < nlon; ilon++)
00196                 for (ilat = 0; ilat < nlat; ilat++)
00197                     tropo_q0[ilon][ilat] = help[ilat * nlon + ilon];
00198         } else
00199             for (ilon = 0; ilon < nlon; ilon++)
00200                 for (ilat = 0; ilat < nlat; ilat++)
00201                     tropo_q0[ilon][ilat] = GSL_NAN;
00202
00203         time1 = times[it + 1];
00204         start[0] = (size_t) it + 1;
00205         NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00206         for (ilon = 0; ilon < nlon; ilon++)
00207             for (ilat = 0; ilat < nlat; ilat++)
00208                 tropo_z1[ilon][ilat] = help[ilat * nlon + ilon];
00209         NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00210         for (ilon = 0; ilon < nlon; ilon++)
00211             for (ilat = 0; ilat < nlat; ilat++)
00212                 tropo_p1[ilon][ilat] = help[ilat * nlon + ilon];
00213         NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00214         for (ilon = 0; ilon < nlon; ilon++)
00215             for (ilat = 0; ilat < nlat; ilat++)
00216                 tropo_t1[ilon][ilat] = help[ilat * nlon + ilon];
00217         if (h2o) {
00218             NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00219             for (ilon = 0; ilon < nlon; ilon++)
00220                 for (ilat = 0; ilat < nlat; ilat++)
00221                     tropo_q1[ilon][ilat] = help[ilat * nlon + ilon];
00222         } else
00223             for (ilon = 0; ilon < nlon; ilon++)
00224                 for (ilat = 0; ilat < nlat; ilat++)
00225                     tropo_q1[ilon][ilat] = GSL_NAN;
00226     }
00227     it_old = it;
00228
00229     /* Interpolate... */

```

```

00230     intpol_tropo_3d(time0, tropo_z0, time1, tropo_z1,
00231                    lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00232                    atm->lat[ip], method, &z0, &z0sig);
00233     intpol_tropo_3d(time0, tropo_p0, time1, tropo_p1,
00234                    lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00235                    atm->lat[ip], method, &p0, &p0sig);
00236     intpol_tropo_3d(time0, tropo_t0, time1, tropo_t1,
00237                    lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00238                    atm->lat[ip], method, &t0, &t0sig);
00239     intpol_tropo_3d(time0, tropo_q0, time1, tropo_q1,
00240                    lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00241                    atm->lat[ip], method, &q0, &q0sig);
00242
00243     /* Write output... */
00244     fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
00245            atm->lon[ip], atm->lat[ip]);
00246     for (iq = 0; iq < ctl.nq; iq++) {
00247         fprintf(out, " ");
00248         fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00249     }
00250     fprintf(out, " %g %g %g %g %g %g %g %g\n",
00251            z0, p0, t0, q0, z0sig, p0sig, t0sig, q0sig);
00252 }
00253
00254 /* Close files... */
00255 fclose(out);
00256 NC(nc_close(ncid));
00257
00258 /* Free... */
00259 free(atm);
00260
00261 return EXIT_SUCCESS;
00262 }
00263
00264 /*****
00265 void intpol_tropo_3d(
00266     double time0,
00267     float array0[EX][EY],
00268     double time1,
00269     float array1[EX][EY],
00270     double lons[EX],
00271     double lats[EY],
00272     size_t nlon,
00273     size_t nlat,
00274     double time,
00275     double lon,
00276     double lat,
00277     int method,
00278     double *var,
00279     double *sigma) {
00280
00281     double aux0, aux1, aux00, aux01, aux10, aux11, mean = 0;
00282
00283     int n = 0;
00284
00285     /* Adjust longitude... */
00286     if (lon < lons[0])
00287         lon += 360;
00288     else if (lon > lons[nlon - 1])
00289         lon -= 360;
00290
00291     /* Get indices... */
00292     int ix = locate_reg(lons, (int) nlon, lon);
00293     int iy = locate_reg(lats, (int) nlat, lat);
00294
00295     /* Calculate standard deviation... */
00296     *sigma = 0;
00297     for (int dx = 0; dx < 2; dx++)
00298         for (int dy = 0; dy < 2; dy++) {
00299             if (isfinite(array0[ix + dx][iy + dy])) {
00300                 mean += array0[ix + dx][iy + dy];
00301                 *sigma += SQR(array0[ix + dx][iy + dy]);
00302                 n++;
00303             }
00304             if (isfinite(array1[ix + dx][iy + dy])) {
00305                 mean += array1[ix + dx][iy + dy];
00306                 *sigma += SQR(array1[ix + dx][iy + dy]);
00307                 n++;
00308             }
00309         }
00310     if (n > 0)
00311         *sigma = sqrt(GSL_MAX(*sigma / n - SQR(mean / n), 0.0));
00312
00313     /* Linear interpolation... */
00314     if (method == 1 && isfinite(array0[ix][iy])
00315         && isfinite(array0[ix][iy + 1])

```

```
00317     && isfinite(array0[ix + 1][iy])
00318     && isfinite(array0[ix + 1][iy + 1])
00319     && isfinite(array1[ix][iy])
00320     && isfinite(array1[ix][iy + 1])
00321     && isfinite(array1[ix + 1][iy])
00322     && isfinite(array1[ix + 1][iy + 1])) {
00323
00324     aux00 = LIN(lons[ix], array0[ix][iy],
00325                lons[ix + 1], array0[ix + 1][iy], lon);
00326     aux01 = LIN(lons[ix], array0[ix][iy + 1],
00327                lons[ix + 1], array0[ix + 1][iy + 1], lon);
00328     aux0 = LIN(lats[iy], aux00, lats[iy + 1], aux01, lat);
00329
00330     aux10 = LIN(lons[ix], array1[ix][iy],
00331                lons[ix + 1], array1[ix + 1][iy], lon);
00332     aux11 = LIN(lons[ix], array1[ix][iy + 1],
00333                lons[ix + 1], array1[ix + 1][iy + 1], lon);
00334     aux1 = LIN(lats[iy], aux10, lats[iy + 1], aux11, lat);
00335
00336     *var = LIN(time0, aux0, time1, aux1, time);
00337 }
00338
00339 /* Nearest neighbor interpolation... */
00340 else {
00341     aux00 = NN(lons[ix], array0[ix][iy],
00342                lons[ix + 1], array0[ix + 1][iy], lon);
00343     aux01 = NN(lons[ix], array0[ix][iy + 1],
00344                lons[ix + 1], array0[ix + 1][iy + 1], lon);
00345     aux0 = NN(lats[iy], aux00, lats[iy + 1], aux01, lat);
00346
00347     aux10 = NN(lons[ix], array1[ix][iy],
00348                lons[ix + 1], array1[ix + 1][iy], lon);
00349     aux11 = NN(lons[ix], array1[ix][iy + 1],
00350                lons[ix + 1], array1[ix + 1][iy + 1], lon);
00351     aux1 = NN(lats[iy], aux10, lats[iy + 1], aux11, lat);
00352
00353     *var = NN(time0, aux0, time1, aux1, time);
00354 }
00355 }
```

## Index

- add\_text\_attribute
  - tropo.c, [342](#)
- atm\_basename
  - ctl\_t, [28](#)
- atm\_conv.c, [44](#)
  - main, [44](#)
- atm\_dist.c, [46](#)
  - main, [46](#)
- atm\_dt\_out
  - ctl\_t, [28](#)
- atm\_filter
  - ctl\_t, [28](#)
- atm\_gpfile
  - ctl\_t, [28](#)
- atm\_init.c, [55](#)
  - main, [55](#)
- atm\_select.c, [59](#)
  - main, [59](#)
- atm\_split.c, [63](#)
  - main, [63](#)
- atm\_stat.c, [67](#)
  - main, [68](#)
- atm\_stride
  - ctl\_t, [28](#)
- atm\_t, [3](#)
  - lat, [4](#)
  - lon, [4](#)
  - np, [3](#)
  - p, [3](#)
  - q, [4](#)
  - time, [3](#)
- atm\_type
  - ctl\_t, [29](#)
- balloon
  - ctl\_t, [25](#)
- cache\_t, [4](#)
  - iso\_n, [6](#)
  - iso\_ps, [6](#)
  - iso\_ts, [6](#)
  - iso\_var, [6](#)
  - tsig, [6](#)
  - up, [5](#)
  - usig, [6](#)
  - vp, [5](#)
  - vsig, [7](#)
  - wp, [5](#)
  - wsig, [7](#)
- cape
  - met\_t, [42](#)
- cart2geo
  - libtrac.c, [86](#)
  - libtrac.h, [203](#)
- check\_finite
  - libtrac.h, [204](#)
- cl
  - met\_t, [41](#)
- clim\_hno3
  - libtrac.c, [86](#)
  - libtrac.h, [204](#)
- clim\_oh
  - libtrac.c, [87](#)
  - libtrac.h, [205](#)
- clim\_tropo
  - libtrac.c, [88](#)
  - libtrac.h, [205](#)
- conv\_cape
  - ctl\_t, [26](#)
- csi\_basename
  - ctl\_t, [29](#)
- csi\_dt\_out
  - ctl\_t, [29](#)
- csi\_lat0
  - ctl\_t, [31](#)
- csi\_lat1
  - ctl\_t, [31](#)
- csi\_lon0
  - ctl\_t, [30](#)
- csi\_lon1
  - ctl\_t, [30](#)
- csi\_modmin
  - ctl\_t, [29](#)
- csi\_nx
  - ctl\_t, [30](#)
- csi\_ny
  - ctl\_t, [31](#)
- csi\_nz
  - ctl\_t, [30](#)
- csi\_obsfile
  - ctl\_t, [29](#)
- csi\_obsmin
  - ctl\_t, [29](#)
- csi\_z0
  - ctl\_t, [30](#)
- csi\_z1
  - ctl\_t, [30](#)
- ctl\_t, [7](#)
  - atm\_basename, [28](#)
  - atm\_dt\_out, [28](#)
  - atm\_filter, [28](#)
  - atm\_gpfile, [28](#)
  - atm\_stride, [28](#)
  - atm\_type, [29](#)
  - balloon, [25](#)
  - conv\_cape, [26](#)
  - csi\_basename, [29](#)
  - csi\_dt\_out, [29](#)
  - csi\_lat0, [31](#)
  - csi\_lat1, [31](#)
  - csi\_lon0, [30](#)

csi\_lon1, 30  
csi\_modmin, 29  
csi\_nx, 30  
csi\_ny, 31  
csi\_nz, 30  
csi\_obsfile, 29  
csi\_obsmin, 29  
csi\_z0, 30  
csi\_z1, 30  
direction, 22  
dry\_depo, 27  
dt\_met, 22  
dt\_mod, 22  
ens\_basename, 35  
grid\_basename, 31  
grid\_dt\_out, 31  
grid\_gpfile, 31  
grid\_lat0, 33  
grid\_lat1, 33  
grid\_lon0, 32  
grid\_lon1, 33  
grid\_nx, 32  
grid\_ny, 33  
grid\_nz, 32  
grid\_sparse, 32  
grid\_z0, 32  
grid\_z1, 32  
isosurf, 25  
met\_detrend, 24  
met\_dp, 23  
met\_dt\_out, 25  
met\_dx, 23  
met\_dy, 23  
met\_geopot\_sx, 24  
met\_geopot\_sy, 24  
met\_np, 24  
met\_p, 24  
met\_sp, 23  
met\_sx, 23  
met\_sy, 23  
met\_tropo, 24  
metbase, 22  
molmass, 26  
nq, 13  
oh\_chem, 27  
prof\_basename, 33  
prof\_lat0, 35  
prof\_lat1, 35  
prof\_lon0, 34  
prof\_lon1, 34  
prof\_nx, 34  
prof\_ny, 35  
prof\_nz, 34  
prof\_obsfile, 33  
prof\_z0, 34  
prof\_z1, 34  
psc\_h2o, 27  
psc\_hno3, 28  
qnt\_cape, 18  
qnt\_cl, 18  
qnt\_ens, 14  
qnt\_format, 13  
qnt\_h2o, 17  
qnt\_h2ot, 16  
qnt\_hno3, 18  
qnt\_iwc, 17  
qnt\_lapse, 20  
qnt\_lwc, 17  
qnt\_m, 14  
qnt\_name, 13  
qnt\_o3, 17  
qnt\_oh, 19  
qnt\_p, 16  
qnt\_pc, 17  
qnt\_pel, 18  
qnt\_plcl, 18  
qnt\_plfc, 18  
qnt\_ps, 14  
qnt\_psat, 19  
qnt\_psice, 19  
qnt\_pt, 15  
qnt\_pv, 21  
qnt\_pw, 19  
qnt\_r, 14  
qnt\_rh, 19  
qnt\_rhice, 20  
qnt\_rho, 14  
qnt\_sh, 19  
qnt\_stat, 21  
qnt\_t, 16  
qnt\_tdew, 21  
qnt\_theta, 20  
qnt\_tice, 21  
qnt\_tnat, 21  
qnt\_ts, 14  
qnt\_tsts, 21  
qnt\_tt, 15  
qnt\_tvirt, 20  
qnt\_u, 16  
qnt\_unit, 13  
qnt\_us, 15  
qnt\_v, 16  
qnt\_vh, 20  
qnt\_vs, 15  
qnt\_vz, 20  
qnt\_w, 17  
qnt\_z, 16  
qnt\_zs, 15  
qnt\_zt, 15  
sample\_basename, 35  
sample\_dx, 36  
sample\_dz, 36  
sample\_obsfile, 35  
species, 26  
stat\_basename, 36  
stat\_lat, 36

- stat\_lon, [36](#)
- stat\_r, [36](#)
- t\_start, [22](#)
- t\_stop, [22](#)
- tdec\_strat, [27](#)
- tdec\_trop, [27](#)
- turb\_dx\_strat, [25](#)
- turb\_dx\_trop, [25](#)
- turb\_dz\_strat, [26](#)
- turb\_dz\_trop, [25](#)
- turb\_mesox, [26](#)
- turb\_mesoz, [26](#)
- wet\_depo, [27](#)
- day2doy
  - libtrac.c, [88](#)
  - libtrac.h, [206](#)
- day2doy.c, [73](#)
  - main, [73](#)
- direction
  - ctl\_t, [22](#)
- doy2day
  - libtrac.c, [89](#)
  - libtrac.h, [206](#)
- doy2day.c, [75](#)
  - main, [75](#)
- dry\_depo
  - ctl\_t, [27](#)
- dt\_met
  - ctl\_t, [22](#)
- dt\_mod
  - ctl\_t, [22](#)
- ens\_basename
  - ctl\_t, [35](#)
- fft\_help
  - met\_spec.c, [291](#)
- geo2cart
  - libtrac.c, [89](#)
  - libtrac.h, [207](#)
- get\_met
  - libtrac.c, [89](#)
  - libtrac.h, [207](#)
- get\_met\_help
  - libtrac.c, [91](#)
  - libtrac.h, [209](#)
- get\_met\_replace
  - libtrac.c, [92](#)
  - libtrac.h, [210](#)
- grid\_basename
  - ctl\_t, [31](#)
- grid\_dt\_out
  - ctl\_t, [31](#)
- grid\_gpfile
  - ctl\_t, [31](#)
- grid\_lat0
  - ctl\_t, [33](#)
- grid\_lat1
  - ctl\_t, [33](#)
- grid\_lon0
  - ctl\_t, [32](#)
- grid\_lon1
  - ctl\_t, [33](#)
- grid\_nx
  - ctl\_t, [32](#)
- grid\_ny
  - ctl\_t, [33](#)
- grid\_nz
  - ctl\_t, [32](#)
- grid\_sparse
  - ctl\_t, [32](#)
- grid\_z0
  - ctl\_t, [32](#)
- grid\_z1
  - ctl\_t, [32](#)
- h2o
  - met\_t, [43](#)
- h2ot
  - met\_t, [41](#)
- intpol\_met\_space\_2d
  - libtrac.c, [93](#)
  - libtrac.h, [211](#)
- intpol\_met\_space\_3d
  - libtrac.c, [92](#)
  - libtrac.h, [210](#)
- intpol\_met\_time\_2d
  - libtrac.c, [95](#)
  - libtrac.h, [213](#)
- intpol\_met\_time\_3d
  - libtrac.c, [94](#)
  - libtrac.h, [212](#)
- intpol\_tropo\_3d
  - tropo\_sample.c, [351](#)
- iso\_n
  - cache\_t, [6](#)
- iso\_ps
  - cache\_t, [6](#)
- iso\_ts
  - cache\_t, [6](#)
- iso\_var
  - cache\_t, [6](#)
- isosurf
  - ctl\_t, [25](#)
- iwc
  - met\_t, [43](#)
- jsec2time
  - libtrac.c, [96](#)
  - libtrac.h, [214](#)
- jsec2time.c, [76](#)
  - main, [76](#)
- lapse.c, [78](#)
  - main, [78](#)

lapse\_rate  
  libtrac.c, 96  
  libtrac.h, 214

lat  
  atm\_t, 4  
  met\_t, 39

libtrac.c, 84  
  cart2geo, 86  
  clim\_hno3, 86  
  clim\_oh, 87  
  clim\_tropo, 88  
  day2doy, 88  
  doy2day, 89  
  geo2cart, 89  
  get\_met, 89  
  get\_met\_help, 91  
  get\_met\_replace, 92  
  intpol\_met\_space\_2d, 93  
  intpol\_met\_space\_3d, 92  
  intpol\_met\_time\_2d, 95  
  intpol\_met\_time\_3d, 94  
  jsec2time, 96  
  lapse\_rate, 96  
  locate\_irr, 97  
  locate\_reg, 97  
  nat\_temperature, 98  
  read\_atm, 98  
  read\_ctl, 100  
  read\_met, 105  
  read\_metCAPE, 107  
  read\_met\_cloud, 108  
  read\_met\_detrend, 109  
  read\_met\_extrapolate, 111  
  read\_met\_geopot, 111  
  read\_met\_grid, 113  
  read\_met\_help\_2d, 115  
  read\_met\_help\_3d, 114  
  read\_met\_levels, 115  
  read\_met\_ml2pl, 116  
  read\_met\_periodic, 117  
  read\_met\_pv, 118  
  read\_met\_sample, 119  
  read\_met\_surface, 121  
  read\_met\_tropo, 121  
  scan\_ctl, 124  
  sedi, 125  
  spline, 125  
  stddev, 126  
  time2jsec, 126  
  timer, 127  
  write\_atm, 127  
  write\_csi, 129  
  write\_ens, 132  
  write\_grid, 134  
  write\_prof, 136  
  write\_sample, 139  
  write\_station, 141

libtrac.h, 201

cart2geo, 203  
check\_finite, 204  
clim\_hno3, 204  
clim\_oh, 205  
clim\_tropo, 205  
day2doy, 206  
doy2day, 206  
geo2cart, 207  
get\_met, 207  
get\_met\_help, 209  
get\_met\_replace, 210  
intpol\_met\_space\_2d, 211  
intpol\_met\_space\_3d, 210  
intpol\_met\_time\_2d, 213  
intpol\_met\_time\_3d, 212  
jsec2time, 214  
lapse\_rate, 214  
locate\_irr, 215  
locate\_reg, 215  
nat\_temperature, 216  
read\_atm, 216  
read\_ctl, 218  
read\_met, 223  
read\_metCAPE, 225  
read\_met\_cloud, 226  
read\_met\_detrend, 227  
read\_met\_extrapolate, 229  
read\_met\_geopot, 229  
read\_met\_grid, 231  
read\_met\_help\_2d, 233  
read\_met\_help\_3d, 232  
read\_met\_levels, 233  
read\_met\_ml2pl, 234  
read\_met\_periodic, 235  
read\_met\_pv, 236  
read\_met\_sample, 237  
read\_met\_surface, 239  
read\_met\_tropo, 239  
scan\_ctl, 242  
sedi, 243  
spline, 243  
stddev, 244  
time2jsec, 244  
timer, 245  
write\_atm, 245  
write\_csi, 247  
write\_ens, 250  
write\_grid, 252  
write\_prof, 254  
write\_sample, 257  
write\_station, 259

locate\_irr  
  libtrac.c, 97  
  libtrac.h, 215

locate\_reg  
  libtrac.c, 97  
  libtrac.h, 215

lon

- atm\_t, [4](#)
- met\_t, [39](#)
- lwc
  - met\_t, [43](#)
- main
  - atm\_conv.c, [44](#)
  - atm\_dist.c, [46](#)
  - atm\_init.c, [55](#)
  - atm\_select.c, [59](#)
  - atm\_split.c, [63](#)
  - atm\_stat.c, [68](#)
  - day2doy.c, [73](#)
  - doy2day.c, [75](#)
  - jsec2time.c, [76](#)
  - lapse.c, [78](#)
  - met\_map.c, [274](#)
  - met\_prof.c, [280](#)
  - met\_sample.c, [286](#)
  - met\_spec.c, [291](#)
  - met\_zm.c, [295](#)
  - time2jsec.c, [301](#)
  - trac.c, [321](#)
  - tropo.c, [343](#)
  - tropo\_sample.c, [353](#)
- met\_detrend
  - ctl\_t, [24](#)
- met\_dp
  - ctl\_t, [23](#)
- met\_dt\_out
  - ctl\_t, [25](#)
- met\_dx
  - ctl\_t, [23](#)
- met\_dy
  - ctl\_t, [23](#)
- met\_geopot\_sx
  - ctl\_t, [24](#)
- met\_geopot\_sy
  - ctl\_t, [24](#)
- met\_map.c, [273](#)
  - main, [274](#)
- met\_np
  - ctl\_t, [24](#)
- met\_p
  - ctl\_t, [24](#)
- met\_prof.c, [280](#)
  - main, [280](#)
- met\_sample.c, [286](#)
  - main, [286](#)
- met\_sp
  - ctl\_t, [23](#)
- met\_spec.c, [290](#)
  - fft\_help, [291](#)
  - main, [291](#)
- met\_sx
  - ctl\_t, [23](#)
- met\_sy
  - ctl\_t, [23](#)
- met\_t, [37](#)
- cape, [42](#)
- cl, [41](#)
- h2o, [43](#)
- h2ot, [41](#)
- iwc, [43](#)
- lat, [39](#)
- lon, [39](#)
- lwc, [43](#)
- np, [39](#)
- nx, [38](#)
- ny, [39](#)
- o3, [43](#)
- p, [39](#)
- pc, [41](#)
- pel, [42](#)
- pl, [44](#)
- plcl, [41](#)
- plfc, [41](#)
- ps, [39](#)
- pt, [40](#)
- pv, [43](#)
- t, [42](#)
- time, [38](#)
- ts, [40](#)
- tt, [40](#)
- u, [42](#)
- us, [40](#)
- v, [42](#)
- vs, [40](#)
- w, [43](#)
- z, [42](#)
- zs, [40](#)
- zt, [41](#)
- met\_tropo
  - ctl\_t, [24](#)
- met\_zm.c, [295](#)
  - main, [295](#)
- metbase
  - ctl\_t, [22](#)
- module\_advection
  - trac.c, [304](#)
- module\_convection
  - trac.c, [305](#)
- module\_decay
  - trac.c, [305](#)
- module\_diffusion\_meso
  - trac.c, [306](#)
- module\_diffusion\_turb
  - trac.c, [308](#)
- module\_dry\_deposition
  - trac.c, [309](#)
- module\_isosurf
  - trac.c, [311](#)
- module\_isosurf\_init
  - trac.c, [310](#)
- module\_meteo
  - trac.c, [312](#)
- module\_oh\_chem



trac.c, 317  
 module\_position  
   trac.c, 314  
 module\_rng  
   trac.c, 315  
 module\_rng\_init  
   trac.c, 315  
 module\_sedi  
   trac.c, 316  
 module\_wet\_deposition  
   trac.c, 318  
 molmass  
   ctl\_t, 26  
  
 nat\_temperature  
   libtrac.c, 98  
   libtrac.h, 216  
 np  
   atm\_t, 3  
   met\_t, 39  
 nq  
   ctl\_t, 13  
 nx  
   met\_t, 38  
 ny  
   met\_t, 39  
  
 o3  
   met\_t, 43  
 oh\_chem  
   ctl\_t, 27  
  
 p  
   atm\_t, 3  
   met\_t, 39  
 pc  
   met\_t, 41  
 pel  
   met\_t, 42  
 pl  
   met\_t, 44  
 plcl  
   met\_t, 41  
 plfc  
   met\_t, 41  
 prof\_basename  
   ctl\_t, 33  
 prof\_lat0  
   ctl\_t, 35  
 prof\_lat1  
   ctl\_t, 35  
 prof\_lon0  
   ctl\_t, 34  
 prof\_lon1  
   ctl\_t, 34  
 prof\_nx  
   ctl\_t, 34  
 prof\_ny  
   ctl\_t, 35  
  
 prof\_nz  
   ctl\_t, 34  
 prof\_obsfile  
   ctl\_t, 33  
 prof\_z0  
   ctl\_t, 34  
 prof\_z1  
   ctl\_t, 34  
 ps  
   met\_t, 39  
 psc\_h2o  
   ctl\_t, 27  
 psc\_hno3  
   ctl\_t, 28  
 pt  
   met\_t, 40  
 pv  
   met\_t, 43  
  
 q  
   atm\_t, 4  
 qnt\_cape  
   ctl\_t, 18  
 qnt\_cl  
   ctl\_t, 18  
 qnt\_ens  
   ctl\_t, 14  
 qnt\_format  
   ctl\_t, 13  
 qnt\_h2o  
   ctl\_t, 17  
 qnt\_h2ot  
   ctl\_t, 16  
 qnt\_hno3  
   ctl\_t, 18  
 qnt\_iwc  
   ctl\_t, 17  
 qnt\_lapse  
   ctl\_t, 20  
 qnt\_lwc  
   ctl\_t, 17  
 qnt\_m  
   ctl\_t, 14  
 qnt\_name  
   ctl\_t, 13  
 qnt\_o3  
   ctl\_t, 17  
 qnt\_oh  
   ctl\_t, 19  
 qnt\_p  
   ctl\_t, 16  
 qnt\_pc  
   ctl\_t, 17  
 qnt\_pel  
   ctl\_t, 18  
 qnt\_plcl  
   ctl\_t, 18  
 qnt\_plfc  
   ctl\_t, 18

qnt\_ps  
    ctl\_t, 14  
qnt\_psat  
    ctl\_t, 19  
qnt\_psice  
    ctl\_t, 19  
qnt\_pt  
    ctl\_t, 15  
qnt\_pv  
    ctl\_t, 21  
qnt\_pw  
    ctl\_t, 19  
qnt\_r  
    ctl\_t, 14  
qnt\_rh  
    ctl\_t, 19  
qnt\_rhice  
    ctl\_t, 20  
qnt\_rho  
    ctl\_t, 14  
qnt\_sh  
    ctl\_t, 19  
qnt\_stat  
    ctl\_t, 21  
qnt\_t  
    ctl\_t, 16  
qnt\_tdew  
    ctl\_t, 21  
qnt\_theta  
    ctl\_t, 20  
qnt\_tice  
    ctl\_t, 21  
qnt\_tnat  
    ctl\_t, 21  
qnt\_ts  
    ctl\_t, 14  
qnt\_tsts  
    ctl\_t, 21  
qnt\_tt  
    ctl\_t, 15  
qnt\_tvirt  
    ctl\_t, 20  
qnt\_u  
    ctl\_t, 16  
qnt\_unit  
    ctl\_t, 13  
qnt\_us  
    ctl\_t, 15  
qnt\_v  
    ctl\_t, 16  
qnt\_vh  
    ctl\_t, 20  
qnt\_vs  
    ctl\_t, 15  
qnt\_vz  
    ctl\_t, 20  
qnt\_w  
    ctl\_t, 17  
qnt\_z  
    ctl\_t, 16  
qnt\_zs  
    ctl\_t, 15  
qnt\_zt  
    ctl\_t, 15  
read\_atm  
    libtrac.c, 98  
    libtrac.h, 216  
read\_ctl  
    libtrac.c, 100  
    libtrac.h, 218  
read\_met  
    libtrac.c, 105  
    libtrac.h, 223  
read\_met\_cape  
    libtrac.c, 107  
    libtrac.h, 225  
read\_met\_cloud  
    libtrac.c, 108  
    libtrac.h, 226  
read\_met\_detrend  
    libtrac.c, 109  
    libtrac.h, 227  
read\_met\_extrapolate  
    libtrac.c, 111  
    libtrac.h, 229  
read\_met\_geopot  
    libtrac.c, 111  
    libtrac.h, 229  
read\_met\_grid  
    libtrac.c, 113  
    libtrac.h, 231  
read\_met\_help\_2d  
    libtrac.c, 115  
    libtrac.h, 233  
read\_met\_help\_3d  
    libtrac.c, 114  
    libtrac.h, 232  
read\_met\_levels  
    libtrac.c, 115  
    libtrac.h, 233  
read\_met\_ml2pl  
    libtrac.c, 116  
    libtrac.h, 234  
read\_met\_periodic  
    libtrac.c, 117  
    libtrac.h, 235  
read\_met\_pv  
    libtrac.c, 118  
    libtrac.h, 236  
read\_met\_sample  
    libtrac.c, 119  
    libtrac.h, 237  
read\_met\_surface  
    libtrac.c, 121  
    libtrac.h, 239  
read\_met\_tropo

- libtrac.c, [121](#)
- libtrac.h, [239](#)
- rng
  - trac.c, [325](#)
- sample\_basename
  - ctl\_t, [35](#)
- sample\_dx
  - ctl\_t, [36](#)
- sample\_dz
  - ctl\_t, [36](#)
- sample\_obsfile
  - ctl\_t, [35](#)
- scan\_ctl
  - libtrac.c, [124](#)
  - libtrac.h, [242](#)
- sedi
  - libtrac.c, [125](#)
  - libtrac.h, [243](#)
- species
  - ctl\_t, [26](#)
- spline
  - libtrac.c, [125](#)
  - libtrac.h, [243](#)
- stat\_basename
  - ctl\_t, [36](#)
- stat\_lat
  - ctl\_t, [36](#)
- stat\_lon
  - ctl\_t, [36](#)
- stat\_r
  - ctl\_t, [36](#)
- stddev
  - libtrac.c, [126](#)
  - libtrac.h, [244](#)
- t
  - met\_t, [42](#)
- t\_start
  - ctl\_t, [22](#)
- t\_stop
  - ctl\_t, [22](#)
- tdec\_strat
  - ctl\_t, [27](#)
- tdec\_trop
  - ctl\_t, [27](#)
- time
  - atm\_t, [3](#)
  - met\_t, [38](#)
- time2jsec
  - libtrac.c, [126](#)
  - libtrac.h, [244](#)
- time2jsec.c, [301](#)
  - main, [301](#)
- timer
  - libtrac.c, [127](#)
  - libtrac.h, [245](#)
- trac.c, [303](#)
  - main, [321](#)
- module\_advection, [304](#)
- module\_convection, [305](#)
- module\_decay, [305](#)
- module\_diffusion\_meso, [306](#)
- module\_diffusion\_turb, [308](#)
- module\_dry\_deposition, [309](#)
- module\_isosurf, [311](#)
- module\_isosurf\_init, [310](#)
- module\_meteo, [312](#)
- module\_oh\_chem, [317](#)
- module\_position, [314](#)
- module\_rng, [315](#)
- module\_rng\_init, [315](#)
- module\_sedi, [316](#)
- module\_wet\_deposition, [318](#)
- rng, [325](#)
- write\_output, [319](#)
- tropo.c, [342](#)
  - add\_text\_attribute, [342](#)
  - main, [343](#)
- tropo\_sample.c, [351](#)
  - intpol\_tropo\_3d, [351](#)
  - main, [353](#)
- ts
  - met\_t, [40](#)
- tsig
  - cache\_t, [6](#)
- tt
  - met\_t, [40](#)
- turb\_dx\_strat
  - ctl\_t, [25](#)
- turb\_dx\_trop
  - ctl\_t, [25](#)
- turb\_dz\_strat
  - ctl\_t, [26](#)
- turb\_dz\_trop
  - ctl\_t, [25](#)
- turb\_mesox
  - ctl\_t, [26](#)
- turb\_mesoz
  - ctl\_t, [26](#)
- u
  - met\_t, [42](#)
- up
  - cache\_t, [5](#)
- us
  - met\_t, [40](#)
- usig
  - cache\_t, [6](#)
- v
  - met\_t, [42](#)
- vp
  - cache\_t, [5](#)
- vs
  - met\_t, [40](#)
- vsig
  - cache\_t, [7](#)

## w

- met\_t, [43](#)

## wet\_depo

- ctl\_t, [27](#)

## wp

- cache\_t, [5](#)

## write\_atm

- libtrac.c, [127](#)

- libtrac.h, [245](#)

## write\_csi

- libtrac.c, [129](#)

- libtrac.h, [247](#)

## write\_ens

- libtrac.c, [132](#)

- libtrac.h, [250](#)

## write\_grid

- libtrac.c, [134](#)

- libtrac.h, [252](#)

## write\_output

- trac.c, [319](#)

## write\_prof

- libtrac.c, [136](#)

- libtrac.h, [254](#)

## write\_sample

- libtrac.c, [139](#)

- libtrac.h, [257](#)

## write\_station

- libtrac.c, [141](#)

- libtrac.h, [259](#)

## wsig

- cache\_t, [7](#)

## z

- met\_t, [42](#)

## zs

- met\_t, [40](#)

## zt

- met\_t, [41](#)